UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

O MEDIADOR DO EPIDEMIC MARKETPLACE

Hugo Miguel Martins Brito Ferreira

Mestrado em Engenharia Informática

Sistemas de Informação

2011

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

O MEDIADOR DO EPIDEMIC MARKETPLACE

Hugo Miguel Martins Brito Ferreira

PROJECTO

Trabalho orientado pelo Prof. Doutor Fabrício Alves Barbosa da Silva
Trabalho co-orientado pelo Prof. Doutor Mário Jorge Costa Gaspar da Silva

Mestrado em Engenharia Informática

Sistemas de Informação

2011

Agradecimentos:

Aos meus Pais. Por todo o esforço e sacrifício feito para que eu chegasse até aqui. Por todo o apoio. Por todo o incentivo. Por serem um exemplo que tento seguir todos os dias. Pela educação que me deram e pelos princípios que me transmitiram. Por terem lutado todos os dias para que eu tivesse as oportunidades que eles não puderam ter.

Ao meu Irmão. Pelas brincadeiras. Por me acordar de manhã. Pela energia e vivacidade que me transmite. Porque é o meu “puto”.

À minha Família. Por me terem tornado na pessoa que sou.

À Tania. Por tudo o que me ensinou. Por tudo o que vivemos. Pelos bons momentos. Pelo amor.

Aos meus Amigos e Colegas de Faculdade. Pela convivência. Pela troca de conhecimentos. Pela união. Pela inter-ajuda. Pelo ambiente saudável, de amizade e de diversão.

Ao Professor Fabrício. Pelo suporte. Pela exigência. Pela orientação. Pela compreensão. Pelos Ensinamentos.

Ao Professor Mário. Por me ter convidado para trabalhar no XLDB. Por ter acreditado em mim. Pelas dicas preciosas. Pelos conselhos. Pela experiência.

À equipa do Projecto Epiwork. Pelo último ano e meio de trabalho. Pela troca de experiências. Pelo apoio. Pelo bom ambiente de trabalho.

Dedicatória:

Para os meus Pais e Irmão.

Para a Tania.

Resumo:

Este relatório apresenta o trabalho centrado na construção de um Mediador para a plataforma Epidemic Marketplace, inserido no projecto europeu Epiwork. O projecto Epiwork visa a criação de um sistema de previsão, detecção e simulação da ocorrência de surtos epidémicos localizados temporal e espacialmente.

A Epidemic Marketplace é a plataforma responsável pela manutenção e gestão dos dados. O Mediador disponibiliza um conjunto de serviços web, através de uma interface RESTful, responsáveis pela mediação entre o Repositório Digital Fedora Commons e os seus utilizadores, possibilitando a estes últimos a concretização de operações de leitura e de escrita sobre o primeiro.

Os serviços de mediação disponibilizados podem ser utilizados directamente pelos Utilizadores autorizados e registados no Epidemic Marketplace, assim como por aplicações por estes concretizadas, desde que as credenciais sejam válidas.

Entre os serviços web implementados há a destacar a concretização da abordagem OAI-ORE no serviço de upload que introduz o conceito de objectos digitais compostos, assim como a integração do protocolo OAI-PMH que consiste na recolha de metadados, no qual se baseia o serviço de pesquisa.

Foi também construído um serviço de transferência de objectos digitais armazenados no repositório. Estando o Repositório Digital assente no Fedora Commons Repository Software, é com a API do Fedora que os serviços de mediação vão sobretudo comunicar de modo a processarem os pedidos.

Por fim, foi construído um esquema de autenticação baseado no método de Acesso Básico de Autorização HTTP, que permite ao Mediador a recepção das credenciais dos Utilizadores para posterior validação das mesmas junto a um servidor LDAP, através da interface JNDI.

Palavras Chave:

Epiwork, Epidemic Marketplace, Mediador, datasets, repositório digital, Fedora Repository, Muradora, OAI-PMH, OAI-ORE, metadados.

Abstract:

This report presents the work on building the Mediator services for the Epidemic Marketplace platform, as part of the European project Epiwork. This project aims to create a system for forecasting, detection and simulation of the occurrence of temporally and spatially localized outbreaks.

The Epidemic Marketplace is responsible for maintaining and managing epidemic data. The Mediator provides a set of web services through a RESTful interface, responsible for the mediation between the Fedora Repository and its users, enabling these to implement operations of reading and writing over the first.

The mediation services provided may be directly used by the registered and authorized users of the Epidemic Marketplace, as well as by any applications built by them, presenting valid credentials.

The implemented upload web service follows OAI-ORE conventions, which support the composite digital objects, while the search web service is based on the OAI-PMH protocol. A download web service for accessing the digital objects stored in the repository was also implemented. Given that the Epidemic Marketplace repository is based on Fedora Commons Repository Software, the mediation services communicate primarily with the Fedora Commons API to process the requests.

Finally, the provided authentication scheme is based on the authentication method for HTTP Basic Access Authorization, which allows the Mediator services to receive the credentials from the users for subsequent validation by an LDAP server accessed through a JNDI interface.

Keywords:

Epiwork, Epidemic Marketplace, Mediador, datasets, digital repository, Fedora Repository, Muradora, OAI-PMH, OAI-ORE, metadata.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1- Introdução

1.1 Contextualização

Nos últimos anos um enorme fluxo de dados quantitativos sociais, demográficos e comportamentais tornaram-se disponíveis devido ao crescimento exponencial da importância e utilização de redes sociais através da internet.

Este enorme fluxo de dados tornou apetecível e motivou a utilização de tecnologias inovadoras que pudessem, de alguma forma, melhorar as capacidades e a precisão dos sistemas de vigilância das doenças tradicionais, proporcionando assim uma melhor e mais eficaz capacidade de detecção e localização do acontecimento de surtos de doença.

O acesso a uma tão grande quantidade e variedade de dados vem trazer o problema do seu armazenamento e categorização, de forma a que os dados obtidos possam posteriormente ser utilizados com utilidade na criação dos já referidos sistemas de vigilância.

O recente aparecimento e mediatização de alguns surtos epidémicos à escala global fez com que estes se tornassem assunto predominante nas redes sociais mais utilizadas, como o Twitter [1], e se revelasse de grande importância e utilidade a criação de sistemas de previsão, detecção e simulação da propagação de epidemias.

É neste contexto que surge o projecto Europeu denominado Epiwork [2], que visa criar um sistema de previsão e detecção de surtos epidémicos localizados temporal e espacialmente. O projecto fornece uma plataforma responsável pela manutenção e gestão dos dados e da informação, que é o Epidemic Marketplace [3]. A ligação entre os diversos componentes desta plataforma e os componentes externos à mesma será da responsabilidade de um Mediador, caracterizado por um conjunto de serviços web.

1.2 Motivação

O Mediador do Epidemic Marketplace é parte integrante do projecto Europeu denominado Epiwork que terá uma duração prevista de 4 anos.

A criação do Mediador pretende vir de encontro às expectativas de fornecer aos utilizadores do Epidemic Marketplace uma ferramenta inovadora, de fácil utilização e orientada à disponibilização de serviços considerados úteis do ponto de vista dos utilizadores e das aplicações que estes possam pretender utilizar, para realizar a interacção com o Mediador.

Os serviços que o Mediador pretende disponibilizar estão direccionados para o acesso aos dados e conteúdos armazenados no repositório do Epidemic Marketplace, através de uma interface RESTful suportada por uma API com a descrição dos serviços e as suas possibilidades de utilização.

Com o desenvolvimento no Mediador, a equipa responsável pela plataforma do Epidemic Marketplace pretende contribuir de forma ainda mais decisiva na construção e no melhoramento de uma plataforma computacional orientada aos dados e à gestão e acesso aos mesmos, inserida no âmbito do projecto Epiwork. Pretende em simultâneo, fomentar a colaboração e o espírito de cooperação e comunicação entre esta equipa e outras pertencentes ao projecto Epiwork, com as quais está mais directamente relacionada.

1.3 O Projecto Epiwork

O projecto Epiwork [2] propõe um esforço de pesquisa multidisciplinar, para o desenvolvimento de um conjunto de ferramentas e conhecimentos necessários à construção de infra-estruturas de previsão de epidemias, posteriormente disponíveis para serem utilizadas por epidemiologistas e cientistas ligados à área da saúde. Este esforço interdisciplinar tem por base as necessidades da investigação epidemiológica, com a participação do consórcio de epidemiologistas, especialistas em saúde pública, biólogos, matemáticos e cientistas de computação.

O projecto Epiwork gira em torno de doze equipas científicas e oito Work Packages (WP) com diferentes responsabilidades. Serão abordados de seguida apenas aqueles que estão mais directamente relacionados com o desenvolvimento da plataforma computacional orientada aos dados, incluindo o WP 3, do qual faz parte o aluno integrado na equipa do LaSIGE (Large-Scale Informatics Systems Laboratory) [4].

O WP 3 (*Information Platform*) será responsável pela criação de uma plataforma de informação, o Epidemic Marketplace. Esta plataforma irá lidar com a dificuldade em assimilar e integrar a variedade crescente dos conjuntos de dados necessários no apoio à abordagem de modelação dos mesmos, assim como a dificuldade em extrair conhecimento e padrões de diferentes fontes de dados. O projecto prevê uma abordagem unificada e integrada para a gestão de recursos através da plataforma Epidemic Marketplace, disponível na web.

O WP 4 (*Computational Modelling Platform*) será responsável pela implementação de uma plataforma de modelação e simulação. Esta plataforma irá integrar modelos, dados reais e técnicas de visualização, de forma a permitir a realização de simulações e facultar o acesso ao estado-da-arte da modelação computacional a um vasto público de especialistas e não especialistas. O objectivo desta plataforma é fornecer uma ferramenta flexível e de fácil manipulação para a simulação de um caso de estudo, teste e validação de um pressuposto específico sobre a propagação de uma doença. Envolve ainda a compreensão de padrões de epidemias observadas, estudo sobre a eficácia e os resultados das diferentes estratégias de intervenção, análise do risco através de cenários do modelo e previsão de novas doenças infecciosas emergentes.

Por fim, o WP 5 (*ICT Monitoring and Reporting System*) terá como responsabilidade o desenvolvimento de um sistema de comunicação e de acompanhamento. Este sistema consiste numa infra-estrutura de monitorização diária que será de grande utilidade para os modeladores, no fornecimento de dados em tempo real para os seus algoritmos de previsão. O sistema será concebido com o objectivo de

informar e educar a população sobre a doença e de colectar em tempo real, informações sobre a saúde da população, através de *web-services*.

1.3.1 O Epidemic Marketplace

O Epidemic Marketplace é a plataforma do sistema a desenvolver no projecto Epiwork responsável pela gestão dos dados e informação.

Os objectivos do Epiwork, que estão directamente relacionados com o funcionamento do Epidemic Marketplace, são os seguintes [5]:

1. Desenvolvimento em grande escala de modelos computacionais orientados aos dados, dotados de um elevado nível de realismo e destinados à previsão do cenário de epidemia.
2. Desenho e implementação de esquemas de recolha de dados originais, motivados pela identificação das necessidades de modelação, tais como a recolha em tempo real de incidências da doença.
3. Instalação de uma plataforma computacional para a investigação epidémica e partilha de dados.

Os principais componentes desta plataforma são [6]:

1. O repositório com os conjuntos de dados e o catálogo das fontes de origem desses mesmos dados, contendo os metadados descritores das bases de dados existentes.
2. O fórum para a publicação da informação relativa aos dados e que promove a colaboração entre os modeladores.

3. O Software de mediação com a capacidade de processar automaticamente as consultas sobre os dados epidemiológicos disponibilizados através das fontes de informação ligadas à plataforma.
4. O Sistema de colecta de dados, responsável por recuperar informações em tempo real de incidências da doença através de fontes de dados públicas, como o são as redes sociais. Após a recolha dos dados, este sistema irá agrupar as incidências de acordo com o seu assunto, criando assim conjuntos de dados (*datasets*) que serão posteriormente armazenados no repositório [7].

Levando em conta o que já foi dito, revela-se de extrema importância e utilidade a criação de uma colecção de Serviços Web que representam o Mediador. Estes funcionarão como interface (fácil de utilizar, flexível e simples – RESTful) de gestão do acesso aos dados contidos no repositório e com capacidades de povoar o repositório, com dados provenientes de fontes e instituições externas ao Epidemic Marketplace.

Como tal, o Mediador do Epidemic Marketplace deve ser responsável por controlar o acesso aos dados provenientes de fontes distintas, relativos a diferentes doenças, disponibilizando-os em diferentes formatos através interfaces quer de pesquisa quer de consulta (através de *queries*), permitindo ainda a autenticação local de utilizadores registados na plataforma.

1.3.2 Implantação Actual do Epidemic Marketplace

No mês de Abril de 2009, os elementos do projecto Epiwork pertencentes ao LaSIGE começaram por montar a estrutura física necessária ao desenvolvimento do Epidemic Marketplace. Foram instalados e configurados dois servidores com o Sistema Operativo CentOS e duas unidades de armazenamento de dados em rede, de modo a conferir a este sistema, um razoável nível de redundância e tornar a recuperação a falhas num processo fiável, fácil e rápido.

Actualmente já se encontra online o site que corresponde ao repositório do Epidemic Marketplace. Este repositório encontra-se baseado no software Fedora Repository [8] e no *front-end* utilizado para trabalhar em cima deste software que é o sistema Islandora.

1.4 Objectivos Específicos

Os objectivos gerais deste projecto são os seguintes:

- Construção do Mediador do Epidemic Marketplace [3], que é a plataforma do Epiwork responsável pela gestão dos dados e da informação.

- Construção de um conjunto de serviços web que suportem o acesso aos dados e conteúdos armazenados no repositório do Epidemic Marketplace e aos componentes externos ao repositório.

- Implementação de uma interface RESTful, que obedeça aos princípios da arquitectura REST [9] e ofereça aos utilizadores uma interacção intuitiva, flexível e simples.

- Integração do protocolo OAI-PMH [10], na consulta e recuperação de metadados, que visa permitir a conteúdos da Web a publicação e arquivamento dos mesmos. A integração deste protocolo será visível no serviço web de pesquisa de objectos digitais armazenados no repositório.

- Implementação da abordagem OAI-ORE [11], um padrão que define normas para descrição e intercâmbio de agregações de recursos web. Estas agregações, podem combinar diferentes tipos de recursos em composições de objectos digitais. A implementação desta abordagem será visível no serviço web de *upload*.

- Construção de um esquema de autenticação baseado no método de Acesso Básico de Autorização HTTP (*Basic Access Authentication HTTP*) através do qual o Mediador faz a recolha das credenciais dos utilizadores para posterior validação junto de um servidor LDAP [12], a partir de uma interface JNDI.

1.5 Contribuições

A realização deste trabalho proporcionou uma contribuição para a comunidade científica na forma da seguinte publicação [13]:

- **Título:** Epidemic Marketplace: An Information Management System for Epidemiological Data
- **Autores:** Luis Filipe Lopes, Fabrício A.B. Silva, Francisco Couto, João Zamite, Hugo Ferreira, Carla Sousa, Mário J. Silva
- **Publicado em:** Proceedings of the ITBAM – DEXA
- **Ano:** 2010

A realização deste trabalho proporcionou ainda a criação de um protótipo do Mediador, que se encontra online desde Outubro de 2010 na página do Epidemic Marketplace.

1.6 Organização do Documento

Este documento encontra-se organizado da seguinte forma:

- **Capítulo 2 – Trabalho Relacionado**

Neste capítulo é feita uma análise de tecnologias e trabalhos relacionados com aquele que é apresentado neste relatório. É um capítulo construído, essencialmente, com base na investigação e estudo de artigos científicos.

- **Capítulo 3 – Mediador - Requisitos Essenciais**

Neste capítulo são detalhados os objectivos inerentes à realização deste projecto, assim como os seus requisitos. São ainda apresentados diagramas e tabelas que pretendem dar uma visão mais estruturada do projecto

- **Capítulo 4 – Implementação**

Neste capítulo são abordadas em detalhe todas as questões relativas à implementação do trabalho. São discutidas e justificadas todas as decisões de implementação tomadas durante o decorrer do projecto e é feita uma reflexão sobre o ganho/custo resultante dessas decisões para o mesmo.

- **Capítulo 5 – Testes de Desempenho e Escalabilidade**

São documentados neste capítulo os testes ao Mediador do Epidemic Marketplace realizados após o fim do seu desenvolvimento, assim como os resultados extraídos desses mesmos testes. É ainda apresentada uma análise ao resultados verificados e analisada a ferramenta JMeter [14] utilizada para a concretização dos testes.

- **Capítulo 6 – Conclusão**

Neste capítulo é abordada a importância da realização deste trabalho, assim como a sua utilidade para o projecto Epiwork. É feito um pequeno resumo do trabalho realizado e apresentadas algumas considerações finais sobre o mesmo.

Capítulo 2 - Trabalho Relacionado

Este capítulo apresenta os trabalhos e tecnologias relacionados com o desenvolvimento do Mediador do Epidemic Marketplace.

O estudo e compreensão de cada um deles, revelou-se decisivo para a realização deste projecto, tendo todos eles contribuído de alguma forma para a tomada de decisões e resolução de problemas que surgiram quer durante a implementação do Mediador, quer durante a escrita deste relatório.

2.1 SOAP vs REST

O desenvolvimento de serviços web está actualmente dividido em duas metodologias, que resultam em duas formas de os pensar e de os implementar:

- A abordagem tradicional e baseada em padrões - **SOAP**;
- A abordagem mais recente e conceptualmente mais simples – **REST**

2.1.1 SOAP

Fornecer uma interoperabilidade contínua entre componentes de tecnologia de *middleware* heterogéneos e promover a independência implementacional (*loose coupling*) entre o consumidor do serviço e o seu fornecedor, são as principais metas do conceito de Arquitectura Orientada aos Serviços (SOA – *Service Oriented Architecture*) e das tecnologias de serviços Web .

Ao nível da tecnologia, SOAP é uma linguagem XML que define a arquitectura e o formato da mensagem. O documento SOAP define um elemento XML chamado *envelope* que contém o cabeçalho (elemento extensível que pode incluir informações relacionadas com o roteamento ou com a Qualidade de Serviço – QoS) e o corpo da mensagem (onde é transportado o conteúdo da mesma). O *XML Schema* é utilizado para descrever a estrutura da mensagem SOAP.

WSDL (*Web Services Description Language*) é uma linguagem XML que define sintacticamente as interfaces utilizadas em SOAP, funciona como uma espécie de contracto entre o fornecedor do serviço e o seu consumidor.

SOAP foi projectado desde o início para ser extensível, de forma a permitir a sua integração com outros padrões normalmente referidos como WS-* (WS-Addressing, WS-Policy, WS-Security, WS-Federation, WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction, WS-RemotePortlets, etc).

Parte da complexidade inerente ao protocolo SOAP advém da multiplicidade de padrões que evoluíram em torno do mesmo.

Na *Figura 1* é possível observar o exemplo da estrutura básica de uma mensagem SOAP.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
        <env:Header>
                <!-- Header information here -->
        </env:Header>
        <env:Body>
                <!-- Body or "Payload" here, a Fault if error happened -->
        </env:Body>
</env:Envelope>
```

Figura 1: Estrutura básica de uma mensagem SOAP

2.1.2 REST

Esta metodologia para o desenvolvimento de sistemas Web tem as suas bases no modelo SOA (*Service-Oriented Architecture*). Muito resumidamente, é um modelo que inter-relaciona as diferentes funcionalidades das aplicações (serviços), através de interfaces e contratos entre estes serviços [15].

REST não é no entanto uma arquitectura, um padrão ou uma tecnologia, mas sim um conjunto de princípios de arquitectura para o desenvolvimento de sistemas Web. O estilo REST pretende dar uma imagem do design de uma aplicação e do modo como esta se comportará: uma rede de Websites, na qual um utilizador progride com uma aplicação, seleccionando as ligações (transições de estado), tendo como resultado a página seguinte, que está a ser transferida para o utilizador e apresentada para o seu uso.

A ideologia do REST reside na necessidade de manter presente a lógica Web do recurso. Um dos conceitos importantes, segundo os princípios da metodologia REST, é a existência de recursos, cada um referenciado por um identificador global, o que será possível através da utilização de URIs (*Uniform Resource Identifier*). De forma a manipular estes recursos, os agentes intervenientes na rede (no caso utilizadores e servidores) devem comunicar via uma interface padronizada (no caso HTTP) e realizar a permuta de representações dos mesmos recursos.

Existem duas perspectivas possíveis para o processo de desenho arquitectural.

Numa delas, o designer começa do zero, construindo uma arquitectura através da interligação de componentes até satisfazer as necessidades do sistema pretendido. Na segunda hipótese, o designer tem, inicialmente, a percepção dos requisitos do sistema como um todo sem restrições, as quais são gradualmente identificadas e aplicadas aos elementos do sistema, levando-o a funcionar como se esperava.

O desenvolvimento do estilo arquitectónico REST segue esta última perspectiva.

Princípios REST:

Uma interface RESTful é baseada em 4 princípios [15]:

- ✓ **Identificação de Recursos através de URI**

Um serviço Web RESTful expõe um conjunto de recursos que são identificados individualmente, nas interacções com os consumidores do serviço. Os recursos são identificados através de URIs.

- ✓ **Manipulação de Recursos através das suas Representações**

Os recursos são conceptualmente separados da sua representação sendo esta última apresentada ao cliente. A representação de um recurso deve ser fiél ao actual estado do mesmo no sistema e deve conter todas as informações necessárias para possibilitar aos clientes a sua manipulação. A manipulação dos recursos é realizada através dos 4 principais métodos do protocolo HTTP:

- GET (acesso);
- POST (criação);
- PUT (alteração);
- DELETE (remoção).

- ✓ **Mensagens auto-descritivas**

A dissociação dos recursos da sua representação possibilita que o acesso ao conteúdo destes possa ser realizado através de uma maior variedade de formatos (HTML, XML, texto, PDF, JPEG, etc.). Cada mensagem deve conter informação suficiente que descreva a forma como deve ser processada. As mensagens de resposta devem indicar de forma explícita se o seu conteúdo é ou não susceptível de ser mantido em cache.

- ✓ **Interacções Stateful através de hiperlinks**

Todas as interacções com os recursos são efectuadas sem aproveitamento do estado (*stateless*), ou seja, a informação que segue na mensagem do pedido é auto-suficiente. Já as interacções com aproveitamento do estado (*stateful)* são baseadas no conceito de transferência de um estado explícito entre interacções, que pode ser concretizada através de várias técnicas como a rescrição do URI, utilização de cookies e de formulários ocultos.

O estado de um recurso pode ser incorporado na mensagem de resposta (através de *hiperlinks*) de modo a possibilitar ao cliente a transição para um futuro estado válido da interacção.

Restrições REST:

O estilo arquitectónico REST deve respeitar as seguintes restrições [9]:

- ✓ **Cliente – Servidor**

Esta restrição deriva do estilo de arquitectura Cliente-Servidor. Ao separar as preocupações relacionadas com a interface de utilizador, das preocupações relacionadas com o armazenamento de dados, a portabilidade da interface através de múltiplas plataformas é melhorada. O mesmo acontece com a escalabilidade do sistema, visto que os componentes do servidor são simplificados, podendo estes evoluir de forma independente.

- ✓ **Comunicação sem estado**

As comunicações devem ser *stateless*, de modo a que cada pedido do cliente ao servidor deva conter todas as informações necessárias para a sua satisfação, não podendo ser aproveitado qualquer contexto armazenado no servidor.

Esta restrição melhora a visibilidade (pois o sistema de controlo consegue determinar com facilidade a natureza de um pedido), a confiabilidade (pois torna mais fácil a recuperação de falhas) e a escalabilidade (pois ao não guardar estado os componentes do servidor libertam os recursos mais rapidamente e a sua implementação é muito menos complexa).

No entanto, esta restrição pode diminuir o desempenho da rede, pois obriga ao envio redundante de dados, para pedidos consecutivos, uma vez que o contexto não pode ser armazenado pelo servidor.

- ✓ **Cache**

Restrição que visa uma maior eficiência da rede.

Esta restrição obriga a que os dados contidos na resposta a um pedido sejam identificados como cacheáveis ou não. Em caso de o serem, a cache do cliente tem a possibilidade de os armazenar e reutilizá-los num pedido equivalente, posteriormente efectuado.

As vantagens prendem-se com a possibilidade de algumas interacções serem desnecessárias, aumentando a eficiência e a escalabilidade do sistema, reduzindo o tempo de resposta ao utilizador. A utilização da cache pode no entanto reduzir a confiabilidade do sistema, caso os dados recebidos através da cache estejam obsoletos, em relação aos dados que seriam recebidos directamente através do servidor.

- ✓ **Interface Uniforme**

A principal característica que distingue o estilo arquitectural REST de outros é a existência de uma interface uniforme entre os seus componentes. Esta restrição visa simplificar a arquitectura do sistema e melhorar a visibilidade das interacções.

Pode implicar uma redução da eficiência do sistema, pois obriga à utilização de uma interface padronizada, ao invés de uma específica que satisfaça melhor as necessidades de uma aplicação.

A interface REST foi projectada para ser eficiente na grande maioria das transferências de dados hipermédia, em especial para o caso da Web. REST é definido por 4 restrições de interface:

Identificação dos recursos; Manipulação dos recursos através de representações; Mensagens auto descritivas e Hipermédia como o “motor” do estado da aplicação.

- ✓ **Sistema em Camadas**

Restrição que visa melhorar o comportamento do sistema em relação às necessidades de escalabilidade da Internet. Restringe o comportamento dos componentes, de modo a que estes apenas possam interagir com a camada imediatamente seguinte. Ao restringir o conhecimento do sistema, limita-se a complexidade da sua implementação.

2.1.3 Vantagens e Desvantagens

Os serviços web SOAP e REST apresentam filosofias muito diferentes. SOAP é um protocolo baseado em XML para computação distribuída. REST tem um design baseado na arquitectura web combinado com características próprias. O protocolo SOAP não é por si só um protocolo muito complexo, no entanto, pode tornar-se num, aquando da utilização das suas numerosas extensões.

Vantagens SOAP:

- Independente da linguagem, plataforma e camada transporte;
- Projectado para lidar com ambientes de computação distribuída;
- Prevalece como o padrão para os serviços web, suportando melhor a integração com outros padrões (WSDL, WS-*);
- Tratamento de erros embutido;
- Extensibilidade.

Desvantagens SOAP:

- Conceptualmente mais difícil e complexo do que o estilo REST;
- Demasiados detalhes;
- Mais difícil de desenvolver, requer ferramentas.
- Envolve uma maior carga passada através das requisição dos pedidos ao serviço

Vantagens REST:

- Independente da linguagem e da plataforma;
- Desenvolvimento muito mais simples em comparação com SOAP
- Curva de aprendizagem reduzida, menor dependência de ferramentas;
- Conciso, não necessita de uma camada adicional de mensagens
- Mais próximo do design e filosofia da Web

- Respostas *human readable*, legíveis e perceptíveis.

Desvantagens REST:

- Assume um modelo de comunicação "ponto-a-ponto", não compatível com o ambiente de computação distribuída, no qual a mensagem pode passar por um ou mais intermediários;
- Carência de padrões de suporte para questões de segurança, política, confiabilidade de mensagens, etc. Logo, serviços mais sofisticados tornam-se mais complexos de desenvolver sob os princípios REST.
- Está dependente do protocolo de transporte HTTP.

2.1.4 Serviços REST e Serviços SOAP

REST é um estilo arquitectónico adequado a aplicações que não requerem segurança para além daquela que já é fornecida pela infraestrutura HTTP, na qual o HTTP é o protocolo apropriado. Flickr, Google Maps e Amazon fornecem serviços web RESTful.

Tanto o eBay como a *Amazon's Simple Storage Service* (S3) fornecem serviços quer em REST quer em SOAP. Os serviços REST são ainda normalmente utilizados por sítios web estáticos e aplicações web *read-only* como motores de busca [16].

Os serviços construídos com base na tecnologia SOAP são normalmente adequados para aplicações com maior complexidade nas suas operações e que requerem segurança sofisticada, confiabilidade ou outros recursos que estejam previstos pelos padrões WS-*. São ainda indicados quando um protocolo de transporte diferente de HTTP tem de ser utilizado. Muitos serviços web da Amazon, principalmente os que envolvem transacções comerciais são baseados em SOAP, assim como os serviços web utilizados por bancos e agências governamentais.

2.2 Autenticação em REST

O estilo de arquitectura REST possibilita a construção de aplicações web escaláveis, no entanto, a sua integração com o actual modelo de segurança e privacidade da web não é pacífica.

Um dos desafios é a impossibilidade do modelo de segurança da web realizar uma boa utilização da técnica de *caching*, quando um número muito grande de utilizadores partilha dados confidenciais em comunidades sociais.

As actuais implementações de segurança incluem a autenticação do utilizador e ainda a utilização de TLS (*Transport Layer Security*), para protecção das sessões de utilizador. Os certificados são utilizados para autenticar os URLs junto dos clientes, mas este esquema delega, de forma errada, no utilizador humano a compreensão dos links e certificados, dando origem ao crescimento dos ataques de *phishing* [17].

O artigo *"Restful Security"* [18] da autoria de Dan Forsberg propõe uma nova forma de tornar seguros os serviços RESTful sem a necessidade da utilização de URLs seguros. A ideia é aumentar a performance dos serviços que requerem protecção da confidencialidade, retirando às redes de armazenamento de dados a responsabilidade de lidar com os requisitos de segurança. Isto é conseguido através da separação entre a decisão de controlo de acesso e o pedido em si.

A solução proposta pelo autor deste artigo consiste na criação de uma chave de protecção de conteúdo, fornecida aos utilizadores autorizados a aceder aos conteúdos cifrados. Todos os conteúdos seriam encriptados com a chave de protecção.

A autenticação dos utilizadores e a decisão de autorização de acesso aos dados seria implementada com preocupações de segurança (como por exemplo através do uso de canais TLS ou através de mecanismos de autenticação e autorização HTTP). Já o acesso aos dados seria realizado sem preocupações de segurança, uma vez que estes só poderiam ser desencriptados através da chave de protecção de conteúdo.

"FOAF + SSL: RESTful Authentication for the Social Web" [19] é o título de um artigo que descreve um protocolo simples de autenticação em aplicações do estilo RESTful, através da utilização de tecnologias amplamente difundidas como HTTP, SSL/TLS e vocabulários da Web Semântica.

Possibilita ao utilizador um sistema de *one-click sign-on* em sítios web, não necessitando este de fornecer um nome de utilizador nem mesmo uma palavra-chave. O resultado seria um mecanismo de autenticação seguro, aberto e distribuído.

O protocolo FOAF+SSL fornece um sistema de autenticação global, flexível e seguro. Utilizando a criptografia de chaves públicas, adquire as vantagens de segurança inerentes a esta tecnologia. Sendo um protocolo RESTful e de boa integração com a Web Semântica, consegue encontrar uma maior quantidade de informação sobre uma entidade através das relações entre os conjuntos de dados. Quando comparado com o modelo de Public Key Infrastructure, o protocolo FOAF+SSL elimina a necessidade de existência de uma hierarquia de autoridades para verificação de uma identidade, tornando-se muito mais flexível.

O OpenID [20] é um sistema de identificação desenvolvido por Brad Fitzpatrick, no qual a identificação do utilizador é dada através de URLs ou XRIs, que podem ser verificados por qualquer servidor que execute o protocolo OpenID.

Neste protocolo, o utilizador regista-se junto de um provedor de identidade (*Identity Provider*), através do qual adquire uma identidade digital. Quando este mesmo utilizador pretende autenticar-se num site que suporte OpenID (*Relaying Party*), apresenta o seu identificador OpenID. O utilizador é então reencaminhado para o seu Provedor de Identidade, onde lhe é pedido que insira a sua palavra-chave. Após isto, as credenciais do utilizador são enviadas pelo seu Provedor, para o site onde se pretende autenticar, finalizando o processo.

O Projecto OpenID [21] encontra-se a emergir como uma solução viável para a infra-estrutura de identidade centrada no utilizador. Prova disto mesmo é o conjunto de sites que actualmente já suportam autenticação através de OpenID:

- Sapo;
- Yahoo;

- Google;
- CNN;
- e muitos outros...

As especificações para serviços Web (WS-*), são exclusivas do protocolo SOAP, fornecendo-lhe capacidades adicionais (segurança, política de transacções, etc).

Na sua tese de Mestrado com o título *"Putting the Web Services Specifications to REST"* [22], Dan R. Olsen III, propõe a extensão do protocolo HTTP com as especificações já mencionadas, de modo a que também os serviços REST pudessem beneficiar das suas capacidades, sem comprometer a sua simplicidade, suprindo assim algumas das suas principais limitações. A solução apresentada tira partido da especificação do protocolo HTTP que permite a criação de novos cabeçalhos através da utilização do prefixo "*X –*".

Os metadados presentes na especificação podem ser incluídos nos cabeçalhos HTTP [*Tabela 1*] ou armazenados num documento referenciado no cabeçalho.

Tabela 1*: Exemplo da inclusão da especificação WS-Security no cabeçalho de um pedido HTTP*.

1	GET /cgi-bin/dan/temp_conv.cgi?tempc=32 HTTP/1.1
2	Host: teton.cs.byu.edu
3	User-Agent: Mozilla/5.0
4	Accept: text/xml ,application/xml, application/xhtml+xml,…
5	Keep-Alive: 300
6	Connection: keep-alive
7	X-WSSecurity: <Security> … </Security>

OAuth (*Open Authorization*) [23] é um protocolo *open-source* que permite que um utilizador (provedor do serviço) partilhe os seus dados pessoais armazenados num determinado site, com um outro utilizador ou entidade (consumidor do serviço), sem que para tal tenha de fornecer o seu nome de utilizador ou palavra-chave.

Este protocolo consiste na partilha de *tokens* entre os utilizadores, em vez de dados confidenciais. Cada *token* concede acesso a um site e recurso específico, durante um determinado período de tempo.

O serviço Amazon S3 (ou *Amazon Simple Storage Service*) [24] fornece aos seus utilizadores uma API REST [25], na qual é utilizado um esquema de HTTP personalizado para efeitos de autenticação. Este esquema é baseado em HMAC (*Hash Message Authentication Code*) [26] com o uso de chaves.

Para autenticar um pedido, o utilizador forma uma *string* concatenando os elementos necessários, utilizando de seguida a sua chave secreta AWS (*Amazon Web Services*) para calcular o HMAC dessa mesma *string.* O resultado é depois passado através do cabeçalho *HTTP Authorization*, juntamente com o pedido [*Figura 2*].

```
GET /photos/puppy.jpg HTTP/1.1
Host: johnsmith.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000

Authorization: AWS 0PN5J17HBGZHT7JJ3X82:frJIUN8DYpKDtOLCwo//yllqDzg=
```

Figura 2: Exemplo de um pedido REST autenticado ao serviço Amazon S3

O Protocolo HTTP/1.0 inclui a especificação para um esquema de Autenticação de Acesso Básico (*Basic Access Authentication*) [27], essencialmente desenhado para permitir que um *web browser,* ou outra aplicação cliente, possa fornecer as suas credenciais aquando da realização de um pedido HTTP.

Antes da transmissão, as credenciais do utilizador são concatenadas numa *String* e codificadas com um algoritmo de Base64. Esta informação é posteriormente colocada no cabeçalho *HTTP Authorization* do pedido.

Este esquema não pode no entanto ser considerado um método seguro tendo em vista a autenticação dos utilizadores, uma vez que as credenciais destes (nome de utilizador e palavra-chave), são transmitidas através da rede como texto em claro.

A única possibilidade de cobrir esta limitação para este esquema de autenticação, passa pela conjugação do mesmo com algum sistema externo de segurança, como por exemplo o protocolo SSL/TLS (*Secure Socket Layer/Transport Layer Security*).

O esquema de Autenticação de Acesso Básico baseia-se num modelo em que o cliente deve autenticar-se com um ID de utilizador e uma palavra-chave para cada domínio. O servidor irá atender a solicitação somente se puder validar as credencias fornecidas pelo cliente.

Não existem, neste esquema, outros parâmetros de autenticação opcionais.

As vantagens da utilização deste esquema de autenticação residem na sua simplicidade e razoável eficiência e segurança, quando combinado com o protocolo SSL/TLS. Outras das vantagens deste esquema tem a ver com a sua popularização o que faz com que seja suportado por todos os *web browsers* mais utilizados e conhecidos.

2.3 OAI-ORE e OAI-PMH

A iniciativa OAI (*Open Archives Initiative*) [28] tem o intuito de construir uma *framework* interoperável para arquivos (repositórios institucionais) que contenham conteúdos digitais (bibliotecas digitais). Esta *framework* permite que os utilizadores (Fornecedores de Serviços) concretizem a recolha de metadados através dos Fornecedores de Dados. Estes metadados são posteriormente utilizados para fornecer serviços de valor acrescentado, frequentemente através da combinação de diferentes conjuntos de dados.

De seguida serão descritos os dois padrões específicos fornecidos pela iniciativa OAI:

- OAI-ORE
- OAI-PMH

2.3.1 ORE

Na era digital, as bibliotecas necessitam de gerir um grande número de objectos. Os objectos digitais, quando comparados com os objectos físicos, possuem a vantagem de serem "ligados" a publicações ou a um qualquer outro tipo de agregado, mantendo ainda assim a possibilidade de serem reutilizados noutros contextos, de forma independente.

A iniciativa OAI (*Open Archives Initiative*) fornece, através do seu trabalho ORE (*Object Reuse and Exchange*), uma *framework* para a gestão de agregações de objectos digitais [29].

A construção de um repositório digital envolve a escolha do software sobre o qual este vai funcionar. No entanto, nem sempre a escolha inicial do software se revela acertada. À medida que o repositório se vai desenvolvendo, também os requisitos e necessidades inerentes ao seu funcionamento podem sofrer alterações levando à necessidade de utilizar software diferente do escolhido inicialmente. O que acontece nestes casos? Muitas vezes, apesar de existir uma boa estratégia de preservação dos dados, a transferência dos mesmos entre repositórios pode tornar-se bastante complicada e levar mesmo à perda de conteúdos. O aparecimento da abordagem OAI-ORE pretende alterar esta situação [11].

A especificação OAI-ORE inclui abordagens para a representação dos objectos digitais, facilitando o acesso e inserção das mesmas de forma independente do *software* no qual o repositório assenta, uniformizando assim a descrição das relações entre os objectos digitais.

Como tal, os repositórios podem ser vistos como interfaces de armazenamento remoto, onde os conteúdos são representados e acedidos através de um mapa de recursos OAI-ORE. Assim, em vez de os repositórios armazenarem e organizarem internamente os conteúdos digitais, utilizando posteriormente os mapas para os divulgarem, os repositórios compreendem os mapas no seu mais baixo nível.

Estes mapas OAI-ORE não necessitam de ser geridos exclusivamente pelo repositório.

Segundo a visão ORE, um repositório é um conjunto de recursos, sendo um item um subconjunto lógico desses recursos (ex: imagem, PDF, etc.), tornado disponível pelo mesmo repositório e ao qual estão associados metadados. ORE está especificamente projectado para descrever a localização e as relações entre esses recursos [30].

No padrão ORE, o termo aplicado a um conjunto de recursos é Agregação, sendo que esta descreve recursos aos quais se aplica o termo Recursos Agregados. De forma a representar uma estrutura hierárquica, as Agregações podem estar elas próprias contidas noutras Agregações. Sendo uma Agregação um conceito abstracto, o protocolo ORE utiliza um Mapa de Recursos para fornecer uma representação concreta da mesma. O protocolo suporta a serialização do mapa em *RDF/XML*, *RDFa* e *Atom XML*.

2.3.2 PMH

OAI-PMH (*Open Archives Initiative - Protocol for Metadata Harvesting*) [10] é um protocolo de recolha de metadados que visa permitir a partilha, publicação e arquivamento de metadados referentes a conteúdos da Web, através de repositórios interoperáveis.

Os metadados expostos têm geralmente uma natureza descritiva e são representados através de formatos de metadados cuja complexidade varia, tais como *Dublin Core* [31] ou *MARCXML* [32].

A definição dos metadados no OAI-PMH pode, contudo, ter um interpretação mais liberal, como é visível no projecto conjunto entre o *Laboratório Nacional de Los Alamos (LANL)* e a *Old Dominion University*. Neste projecto o OAI-PMH é utilizado com o intuito de associar aos recursos digitais informações sobre o historial da utilização dos mesmos. Os metadados mantinham assim um registo sumário do histórico de acesso a um recurso específico [33].

A OAI desenvolve e promove padrões de interoperabilidade que visam facilitar a eficiente disseminação de conteúdos. Este protocolo suporta a interoperabilidade através

de um modelo bipartido relativamente simples. De um lado, os provedores de dados (*data providers*) empregam o OAI-PMH para expor os seus dados estruturados e metadados, de várias formas. Por outro lado, os prestadores de serviços (*service providers*) utilizam o OAI-PMH para concretizar a recolha dos metadados e posteriormente processá-los, para que possam trabalhar sobre os mesmos.

O processamento deste protocolo é realizado através de requisições, nas quais podem ser utilizados os seguintes verbos:

- ***Identify*.** São indicadas as principais informações sobre o repositório, como o nome, o identificador, o email do administrador, etc.
- ***ListMetadataFormats*.** Lista os formatos de metadados implementados pelo sistema do repositório. É obrigatória a implementação, pelo menos, do formato Dublin Core.
- ***ListSets*.** Este verbo é usado para recuperar a estrutura de conjunto de um repositório, útil para a recolha selectiva.
- ***ListIdentifiers*.** Lista todos os identificadores de registos do repositório.
- ***ListRecords*.** Lista todos os registos do repositório.
- ***GetRecord*.** Dado um identificador, lista o registo correspondente.

Um fornecedor OAI-PMH pode ser consultado através da utilização de uma variedade de parâmetros, possibilitando assim uma recolha mais selectiva dos seus conteúdos. Uma recolha efectuada com base no protocolo OAI-PMH pode restringir a pesquisa a uma determinada data, filtrando assim apenas os novos conteúdos, tal como pode ser restrita a um determinado conjunto dos mesmos ou a um formato de metadados.

2.3.3 Caso de Uso

O OAI-ORE define assim padrões e recomendações para a descrição e troca de conjuntos de recursos digitais. Quando utilizado de forma combinada com um protocolo de descoberta como o OAI-PMH, fornece a capacidade de replicar todo o conteúdo de um repositório, de modo completamente automático.

Foi através desta ideia base que um conjunto de universidades do estado do Texas criou o projecto intitulado *Texas Digital Library* (TDL), que tem o intuito de adicionar a um repositório Dspace [34], as funcionalidades de recolha de metadados e de disseminação dos conteúdos, através do protocolo PMH e da abordagem ORE, respectivamente [30].

2.4 Fedora Repository

Fedora Repository [8] oferece software de código aberto para garantir a durabilidade e integridade dos conteúdos digitais. Utiliza a semântica para contextualizar e inter-relacionar conteúdos originários de diferentes fontes e para permitir a criação de espaços inovadores de informação colaborativa.

A arquitectura Fedora representa um conjunto extensível de ferramentas para o armazenamento, gestão e disseminação de objectos digitais e das relações entre estes. Fedora acomoda a agregação de conteúdos locais e distribuídos em objectos digitais e a associação de serviços web a estes mesmos objectos.

A popularidade do Fedora é em grande parte devida à sua escalabilidade, assim como à sua flexibilidade na manipulação e gestão de dados heterogéneos. A dissociação que concretiza entre o software do repositório digital e a interface gráfica a operar em cima do mesmo motivou o uso do Fedora em muitos ambientes institucionais. Desta forma, as instituições tinham a liberdade de escolher a interface gráfica personalizada tendo em conta as suas necessidades específicas.

Nota: O sistema de Repositório Fedora utilizado na realização deste projecto, corresponde à versão 2.2.2. Como tal, todas as referências ao sistema Fedora Repository são respeitantes a essa mesma versão.

2.4.1 Modelo do Objecto Digital Fedora

O Fedora define um modelo genérico para os seus objectos digitais, que pode ser utilizado para representar vários tipos de objectos incluindo documentos, imagens, livros electrónicos, conjuntos de dados, metadados, e muitos outros. Este modelo suporta a agregação de um ou mais conteúdos num único Objecto Digital.

Os conteúdos podem ser de qualquer tipo de *mídia* e podem ser armazenados localmente no repositório ou armazenados externamente e referenciados pelo Objecto Digital.

O modelo é simples e flexível, de modo a que muitos tipos diferentes de objectos digitais possam ser criados. A natureza genérica do modelo Fedora permite que todos os objectos sejam geridos de uma forma coerente num repositório Fedora.

O modelo do Objeto Digital Fedora é definido no esquema de linguagem XML.

O esquema *Fedora Object XML* (FOXML) fornece uma representação completa do modelo de objecto digital do Fedora. Este esquema é um formato XML simples que expressa o modelo do objecto digital. A partir da versão 2.0 do Fedora, os objectos digitais serão armazenados internamente no repositório Fedora no formato FOXML.

Este formato pode também ser utilizado para concretizar a inserção de objectos nos repositórios Fedora, bem como a sua extracção.

Um Objecto Digital Fedora é composto por quatro componentes básicos:

- ✓ **Identificador do Objecto Digital (*Digital Object Identifier*):** um identificador único e persistente, para o objecto;

- ✓ **Propriedades do Objecto (Object Properties):** Um conjunto de propriedades descritivas definidas pelo sistema, necessárias para gerir e rastrear o objecto no repositório.

  Os metadados FOXML para um Fedora Digital Object são os metadados que devem ser registados com todos os objectos de forma a facilitar a sua gestão.

Os metadados FOXML são distintos dos restantes metadados armazenados no objecto digital como conteúdos. Este tipo de metadados é requerido pela arquitectura do repositório Fedora. Todos os restantes metadados (metadados descritivos, metadados técnicos) são considerados opcionais e tratados como Datastreams do objecto digital.

As propriedades do objecto descrevem o seu tipo, estado, modelo de conteúdo a que está vinculado, a data criação e de modificação do objecto e o seu rótulo.

✓ **Fluxos de Dados (*Datastreams*):** São os componentes que representam o conteúdo do objecto Fedora. Um objecto pode ter um ou mais Datastreams. O Datastream pode ser relativo a dados ou metadados, e esse conteúdo pode ser armazenado internamente no repositório do Fedora, ou armazenados remotamente (caso em que o Fedora possui um ponteiro para o conteúdo sob a forma de um URL).

Cada objecto tem um Datastream de metadados Dublin Core como padrão.

✓ **Divulgadores (*Disseminators*):** São os componentes de um Fedora DigitalObject que associam um serviço externo ao objecto com a finalidade de proporcionar vistas extensíveis do objecto ou do seu conteúdo, fluxo de dados. Um objecto pode ter zero ou mais divulgadores.

A *Figura 3* fornece uma visualização gráfica do Modelo do Objecto Digital Fedora, auxiliando a sua percepção.

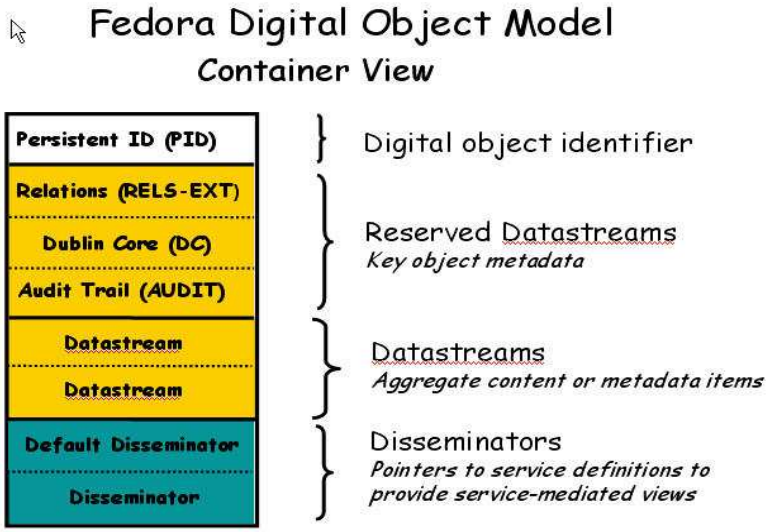


Figura 3: Diagrama do Modelo do Objecto Digital Fedora

Tipos de Objectos Digitais Fedora:

Embora todos os Objectos Fedora respeitem o modelo acima descrito, existem três tipos distintos de objectos digitais que podem ser armazenados num repositório Fedora. A distinção destes três tipos de objectos é fundamental para o funcionamento do sistema de repositório.

Os três tipos de Objectos Digitais Fedora são os seguintes:

- **Objecto de Dados (*Data Objects*):** É o tipo de objectos utilizados para representar uma entidade de conteúdo digital. Estes podem representar um vasto conjunto de entidades tais como imagens, livros, textos electrónicos, publicações, conjuntos de dados, etc. Um ou mais Datastreams representam as partes da entidade de conteúdo digital e um ou mais Disseminadores representam os serviços que podem apresentar diferentes visualizações ou transformações para o mesmo conteúdo.

Os dois seguintes objectos Fedora apresentados são considerados objectos especiais utilizados como blocos de construção para os Disseminadores.

- **Objecto de Definição Comportamental (*Behavior Definition Object*):** É um tipo especial de objecto de controlo utilizado para armazenar uma definição de serviço abstracta, na forma de um conjunto de métodos abstractos. Um Disseminador aponta para um objecto deste tipo como forma de enumerar o conjunto de métodos que suporta. De forma resumida, este objecto define um "contracto comportamental" que um ou mais Objecto de Dados pode "subscrever".

- **Objecto de Mecanismo Comportamental (*Behavior Mechanism Object*):** É um tipo especial de objecto de controlo utilizado para armazenar os metadados vinculativos a serviços concretos. Um Disseminador aponta para um objecto deste tipo como forma de indicar que utiliza a implementação daqueles serviços concretos para correr os métodos do seu serviço.

  Um Objecto de Mecanismo Comportamental está relacionado com um Objecto de Definição Comportamental no sentido em que o primeiro define uma implementação concreta dos métodos abstractos definidos pelo segundo.

2.4.2 Fedora Web Services Interfaces

O sistema Fedora Repository disponibiliza um conjunto de interfaces importantes na construção de Web Services para acesso e gestão dos conteúdos do Repositório Fedora, cuja utilidade para o desenvolvimento dos serviços de mediação do Epidemic Marketplace é inegável.

- ***API-A – Fedora Access Service (SOAP):***

Tal como o nome indica, este serviço define uma interface que possibilita o acesso aos objectos digitais armazenados no repositório. As operações de acesso disponibilizadas por esta interface incluem métodos para descobrir os tipos de disseminações que estão disponíveis para um determinado objecto, assim como para

concretizar o pedido das mesmas. A principal função deste serviço é satisfazer o pedido realizado por um cliente a uma disseminação.

- ***API-A-Lite – Fedora Access Service (REST):***

Esta interface apresenta uma definição de serviços orientados a URIs, para o acesso ao repositório Fedora. Destina-se a suportar acessos que seguem o estilo REST (em contraste com a definição tradicional de serviços web SOAP). Esta interface permite a definição de um URL que pode ser utilizado para realizar solicitações ao serviço de acesso, no entanto, não disponibiliza todas as operações definidas na interface API-A.

- ***API-M – Fedora Management Service (SOAP):***

Este serviço define uma interface para a administração/gestão do repositório, possibilitando a criação, modificação e eliminação de objectos digitais, ou de componentes pertencentes aos mesmos.

- ***API-M-Lite - Fedora Management Service (REST):***

À semelhança da interface API-A-LITE, esta interface apresenta uma definição de serviços orientados a URIs, para a administração/gestão do repositório Fedora. Destina-se a suportar acessos que seguem o estilo REST (em contraste com a definição tradicional de serviços web SOAP). Permite a definição de um URL para realizar solicitações ao serviço de gestão, no entanto, não disponibiliza todas as operações definidas na interface API-A.

- ***Fedora Search Interface:***

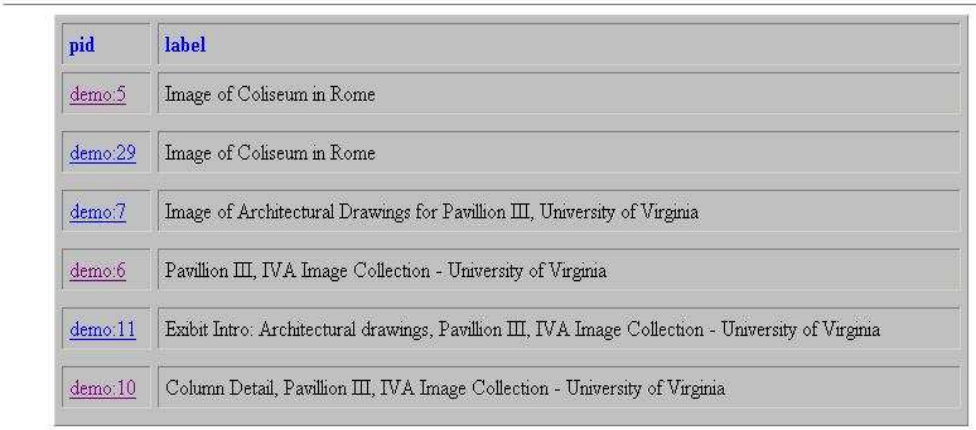
A interface de pesquisa Fedora fornece um mecanismo para busca e navegação do repositório Fedora. Após a inserção de um objecto digital no repositório, os metadados de sistema e os metadados Dublin Core (DC) associados ao objecto, são indexados a uma base de dados relacional, possibilitando a sua pesquisa através desta interface.

O Datastream correspondente aos metadados DC associados a um objecto é opcional. Se nenhum Datastream deste género for fornecido aquando da criação do objecto, o sistema Fedora constrói automaticamente um Datastream DC minimalista que contém apenas dois elementos:

- **dc:identifier –** Contém o valor do identificador do objecto no sistema;
- **dc:title –** Contém o valor do título (caso definido) do objecto.

A interface proporciona pesquisas simples e avançadas em simultâneo, através de um formulário fornecido com o *software* do repositório. Todas as consultas são *case insensitive.* A pesquisa simples permite consultas de palavras e frases que ocorram em qualquer lugar nos campos do objecto de metadados indexados. A pesquisa avançada pode ser realizada sobre uma qualquer combinação de elementos de metadados, utilizando operadores de comparação tanto para campos de texto "*string*" (= e ~), como para campos de data (=,>, ≥, <, ≤).

Os resultados de uma pesquisa através desta interface podem ser obtidos no formato HTML (*Figura 4)* ou XML.



pid	label
demo:5	Image of Coliseum in Rome
demo:29	Image of Coliseum in Rome
demo:7	Image of Architectural Drawings for Pavillion III, University of Virginia
demo:6	Pavillion III, IVA Image Collection - University of Virginia
demo:11	Exibit Intro: Architectural drawings, Pavillion III, IVA Image Collection - University of Virginia
demo:10	Column Detail, Pavillion III, IVA Image Collection - University of Virginia

Figura 4: Exemplo do resultado HTML de uma pesquisa utilizando a Fedora Search Interface

- ***Fedora OAI Provider Interface:***

Como já foi abordado anteriormente neste mesmo capítulo, o protocolo OAI-PMH é um padrão para a partilha e divulgação de metadados entre repositórios. Todos os objectos digitais Fedora possuem um registo primário de metadados Dublin Core, os quais podem ser obtidos através desta interface.

A interface utiliza a sintaxe padrão do protocolo OAI-PMH e processa todas as solicitações OAI-PMH (v2.0) válidas. Está implementada como sendo uma *Java Servlet* (classe Java em conformidade com o *Java Servlet* API, que é um protocolo através do qual uma classe Java pode responder a pedidos HTTP).

A sintaxe para realizar solicitações a esta interface é a seguinte:

http://hostname:port/fedora/oai?verb=oai-pmh2.0-verb

Onde “oai-pmh2.0-verb” será substituído por um dos 6 verbos válidos no protocolo OAI-PMH v2.0.

Capítulo 3 - O Mediador : Especificação

O Mediador do Epidemic Marketplace pode ser considerado um conjunto de serviços web que suportam o acesso aos conteúdos armazenados no repositório do Epidemic Marketplace e às fontes externas de recolha de dados, com base num catálogo descritor das bases de dados de epidemias existentes.

3.1 Requisitos

À semelhança do que acontece com a construção de qualquer sistema ou software, também para a construção do Mediador do Epidemic Marketplace foi realizado um levantamento de requisitos funcionais (que identificam as funcionalidades que o sistema deve implementar) e não funcionais (onde são identificados os restantes requisitos da aplicação e que são considerados requisitos não funcionais uma vez que não estão directamente ligados às funcionalidades do sistema, mas sim ao modo como o sistema se deve comportar).

Nesta secção é então feita a análise aos requisitos inerentes à construção do Mediador do Epidemic Marketplace.

3.1.1 Requisitos Funcionais

- ***Implementação de uma interface RESTful*** [9]**.**

  O Mediador será caracterizado por uma interface que segue os princípios de arquitectura REST, oferecendo aos seus utilizadores uma navegação bastante simples e intuitiva.

  Esta interface dará aos utilizadores do Mediador a possibilidade de interagirem com o repositório da plataforma Epidemic Marketplace. Essa interacção pode ser concretizada em diversos níveis, desde a pesquisa e consulta dos conteúdos e correspondente informação, download destes mesmos

conteúdos e/ou de metatados que os descrevem, upload de recursos e correspondentes metadados de diferentes tipos/formatos.

- ***Implementação de um esquema de Autenticação.***

A utilização dos serviços web fornecidos pelo Mediador do Epidemic Marketplace é exclusiva aos utilizadores registados da plataforma. Como tal, revela-se necessário a utilização de um esquema de autenticação que possa ser chamado a cada pedido aos serviços, uma vez que estamos a lidar com uma interface RESTful, onde não há preservação do estado de autenticação dos utilizadores.

- ***Implementação da capacidade de pesquisa e consulta de conjuntos de dados heterogéneos.***

O Mediador deve ser responsável pela gestão do acesso aos dados heterogéneos que populam o repositório e que posteriormente serão utilizados pela comunidade científica para a realização de simulações, estudos e previsões de surtos epidemiológicos.

Os utilizadores devem ter a possibilidade de pesquisar conteúdos realizando consultas sobre os seus metadados. As consultas podem ser de carácter simples ou composto, caso incidam sobre um único campo de metadados ou mais do que um, respectivamente (e.g. Encontrar conteúdos de determinado autor, e/ou publicados numa determinada data).

- ***Integração do Protocolo OAI-PMH*** [10] ***no serviço de Download***

Todos os conteúdos armazenados no repositório da plataforma Epidemic Marketplace encontram-se devidamente categorizados e descritos pelos correspondentes metadados a eles indexados. O conjunto dos metadados de todos os conteúdos existentes no repositório fornece ao utilizador uma espécie de “catálogo” dos recursos a que pode aceder. Como tal, e visto que os metadados podem ter um tamanho bastante reduzido quando comparados com o

tamanho dos dados, a obtenção desses metadados de forma rápida e útil acaba por se tornar uma funcionalidade importante no Mediador, melhorando e acelerando o processo de selecção dos dados que realmente importam ao utilizador.

Com a implementação deste protocolo, os utilizadores do Mediador podem então realizar a recuperação e visualização de um conjunto de metadados contendo informação e descrição dos conteúdos a que se encontram indexados, de modo a poderem seleccionar os recursos que pretendem obter de forma mais refinada.

- ***Implementação da abordagem OAI-ORE*** [11] ***no serviço de Upload***

Da forma que se encontra estruturada a informação armazenada no repositório da plataforma, é de todo interesse que a inserção de recursos neste mesmo repositório por parte dos utilizadores seja concretizada de maneira a que a integridade da estrutura seja preservada. Como tal, ao permitir aos utilizadores da plataforma efectuar o upload de conteúdos para o repositório, é fundamental permitir e fomentar que estes associem aos seus conteúdos metadados que os descrevam.

A implementação da abordagem OAI-ORE, visa ainda permitir que os utilizadores realizem o upload de objectos digitais compostos, ou seja, que associem a um mesmo recurso, ficheiros de diferentes tipos/formatos.

Com a implementação do serviço de upload pretende dar-se aos utilizadores a possibilidade de introduzirem no repositório, em cada chamada ao serviço, não um conteúdo e o seu ficheiro de metadados correspondente, mas sim um conjunto de conteúdos e de ficheiros de metadados em simultâneo.

3.1.2 Requisitos Não Funcionais

- ***Privacidade e Segurança:***

Sendo a plataforma Epidemic Marketplace essencialmente constituída por um repositório digital onde são armazenados dados epidemiológicos, médicos e sociais, entre outros, que implicam restrições de acesso e privacidade aos mesmos, é essencial que exista um esquema de autenticação que possa ser utilizado para assegurar o acesso autorizado aos serviços web fornecidos pelo Mediador.

Para além deste esquema de autenticação é também conveniente que as operações são realizadas através de conexões seguras.

- ***Simplicidade de Utilização:***

Pretende-se que os serviços web fornecidos pelo Mediador se caracterizem pela sua simplicidade e facilidade de utilização. Deve ser evitado o uso de protocolos ou padrões que requerem demasiada burocracia e uma curva de aprendizagem acentuada.

A ideia é motivar os utilizadores registados na plataforma a construírem aplicações para chamar este serviços web e que ao fazê-lo, se sintam confortáveis.

- ***Legibilidade dos Serviços:***

Pretende-se que ao utilizarem os serviços, os percebam verdadeiramente os serviços que estão a chamar e o que podem obter do processamento dos mesmos. Os serviços devem ter uma linguagem perfeitamente legível e que sintacticamente faça sentido.

Para garantir este requisito, a utilização de URIs pode ser uma boa política.

3.2 Diagramas

3.2.1 Diagrama de Blocos Estrutural do Mediador

É apresentado na *Figura 5* um diagrama que fornece uma visão geral das interacções, nas quais o Mediador é o interveniente principal.

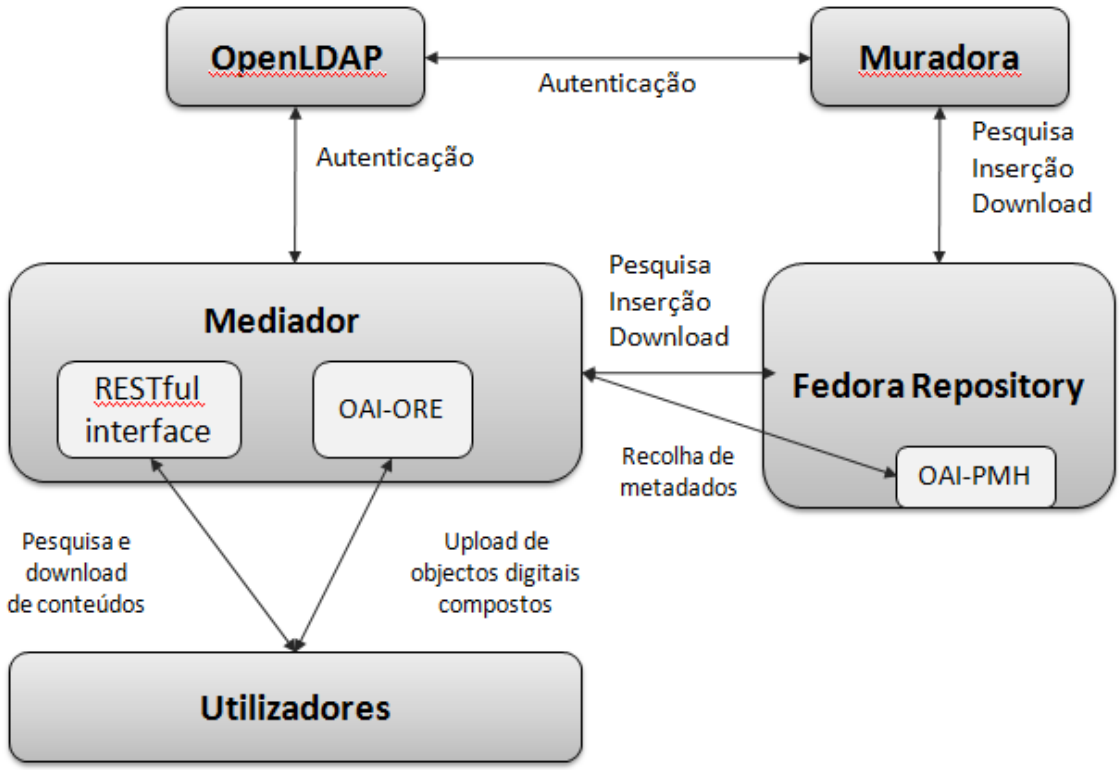


Figura 5: Diagrama de Blocos Estrutural do Mediador

Como exemplificado na imagem, o Mediador vai verificar a validade da autenticação dos seus utilizadores através da ligação ao servidor OpenLDAP [35].

O Mediador irá realizar as operações de consulta e actualização dos conteúdos armazenados no repositório Fedora, através dos métodos disponibilizados pelas API do mesmo.

O Mediador vai ainda concretizar a implementação de uma Interface RESTful e da abordagem OAI-ORE. Simultaneamente vai realizar a integração do Procolo OAI-

PMH que é implementado pelo repositório Fedora e disponibilizado através da *Fedora OAI Provider Interface* . São estes os componentes responsáveis pela interacção entre o Mediador e os seus utilizadores. Através da integração protocolo OAI-PMH, os utilizadores terão a capacidade de concretizar a recuperação de metadados descritores dos conteúdos existentes no repositório.

A interface RESTful permitirá aos utilizadores realizar operações de consulta, pesquisa e download de conteúdos. Por fim, a implementação da abordagem OAI-ORE irá atribuir à funcionalidade de upload, a capacidade de inserção de objectos digitais compostos por parte dos utilizadores, no repositório digital.

3.2.2 Diagramas de Interacções – Serviços Web

Nesta secção serão apresentados 3 diagramas de interacção, um para cada serviço web disponibilizado pelo Mediador, mais concretamente:

1. Serviço Web de Pesquisa;
2. Serviço Web de Download;
3. Serviço Web de Upload.

Cada diagrama representa um exemplo possível de uma sequência de interacções nas quais surgem os diferentes intervenientes no processamento do serviço.

Em qualquer um deles foi tido em consideração que todas as acções seriam executadas com sucesso na primeira tentativa, ou seja, que o Utilizador iria fornecer correctamente o seu nome de utilizador e palavra chave, que iria realizar uma pesquisa sobre parâmetros que são válidos, que iria requisitar um conteúdo através de um ID que de facto existe, etc.

1. Serviço Web de Pesquisa:

Na *Figura 6* encontra-se o exemplo de um possível diagrama de interacções para o serviço web de pesquisa do Mediador.

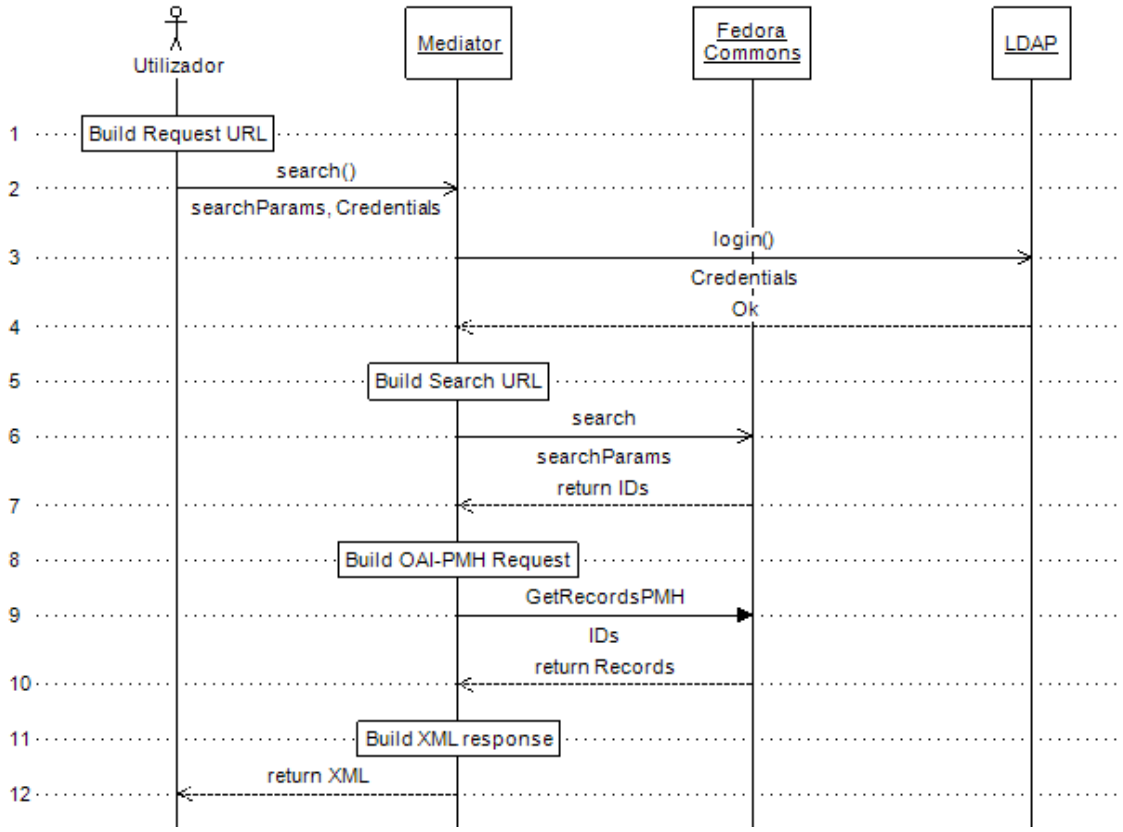


Figura 6: Diagrama de interacções – Serviço de Pesquisa

Neste diagrama, o primeiro interveniente é o Utilizador que utiliza a API dos serviços do Mediador, para construir o URL do pedido, adicionando os parâmetros de pesquisa e respectivos valores. O Utilizador faz então a chamada ao serviço enviando também as suas credenciais do Epidemic Marketplace. O Mediador utiliza estas credenciais para verificar a autenticação deste utilizador junto do servidor LDAP.

Após confirmar que está na presença de um Utilizador autenticado, o Mediador realiza uma chamada à Interface de Pesquisa Fedora, enviando como atributos os parâmetros e respectivos valores para a pesquisa. Como resultado, a Interface devolve um XML com um conjunto de IDs que correspondem aos objectos digitais aramazenados no repositório que satisfazem os critérios da pesquisa.

Para cada um dos IDs recebidos, o Mediador constrói um pedido enviado à Interface OAI Fedora, utilizando o verbo *GetRecord* para obter o registo de Metadados Dublin Core associado aos objectos digitais resultantes da pesquisa.

Posteriormente, o Mediador agrega num único ficheiro XML com a sintaxe OAI-PMH, todos os registos obtidos enviando de seguida o ficheiro para o Utilizador, como resposta ao seu pedido inicial.

2. Serviço Web de Download:

Na *Figura 7* é visível o exemplo de um possível diagrama de interacções para o serviço web de *download* do Mediador.

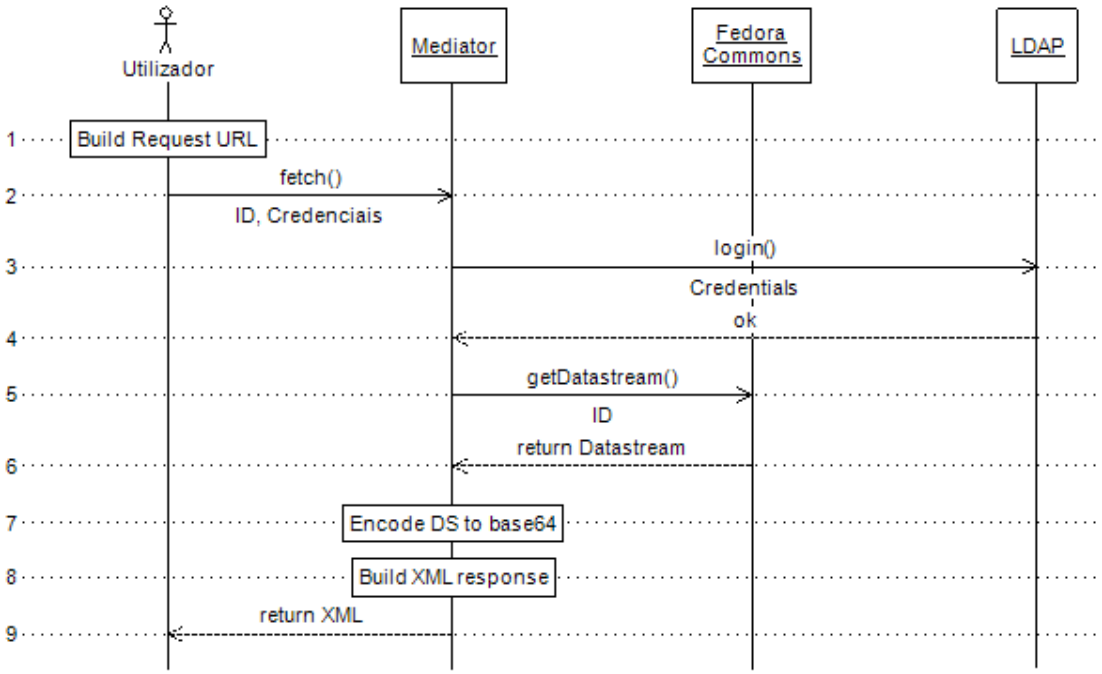


Figura 7: Diagrama de interacções – Serviço de Download

Mais uma vez, à semelhança do diagrama anterior, é ao Utilizador que cabe a responsabilidade de despoletar a acção, construindo o URL do pedido de download e enviando-o junto com as suas credenciais do Epidemic Marketplace, para o Mediador.

Por sua vez o Mediador verifica junto do servidor LDAP a autenticação do Utilizador, através das suas credenciais. Após concluída a autenticação o Mediador realiza um pedido à Interface *API-A Lite – Fedora Acess Service*, enviando como atributo o ID do conteúdo digital pretendido pelo utilizador, obtendo assim a disseminação do mesmo conteúdo.

A disseminação obtida é então codificada pelo Mediador em *Base64* e embutida no XML de resposta que o Mediador envia para o Utilizador, como é visível na última acção deste diagrama.

3. Serviço Web de Upload:

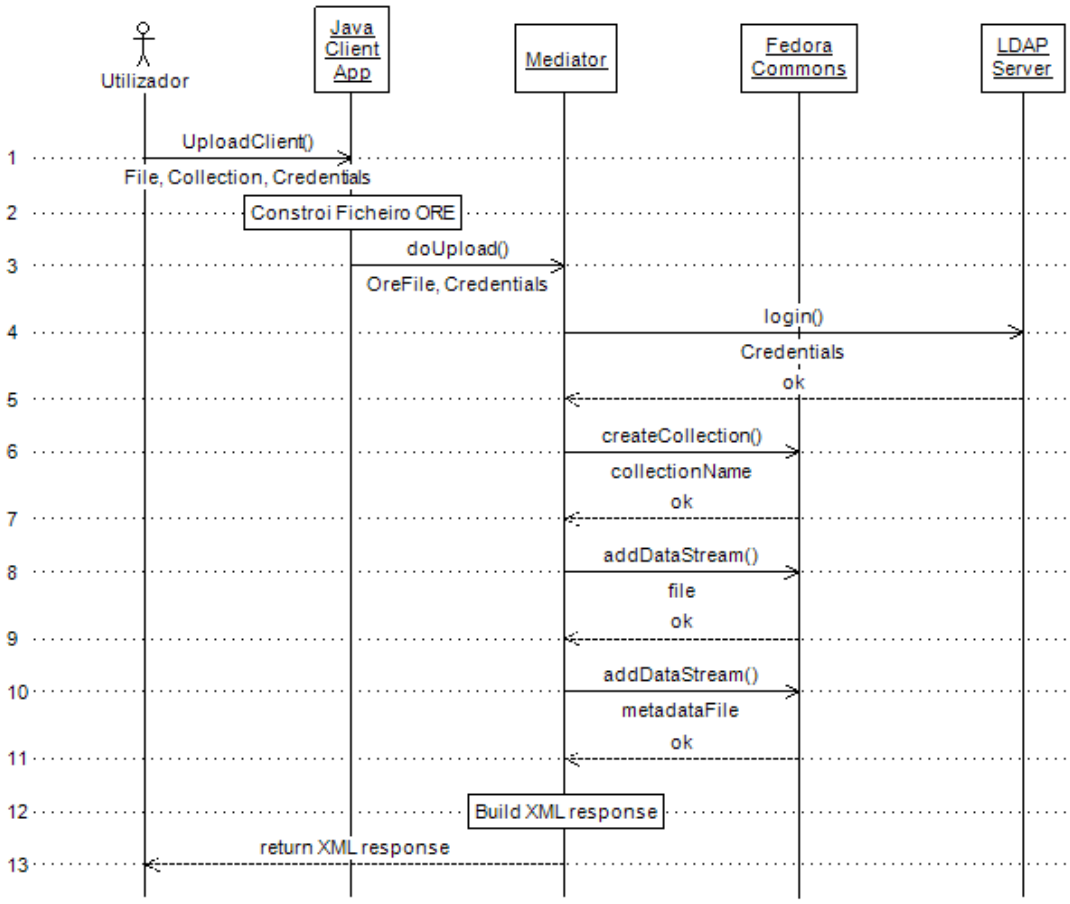


Figura 8: Diagrama de interacções – Serviço de Upload

O Utilizador realiza uma chamada à Aplicação Cliente Java fornecida pelo Mediador, na qual fornece como parâmetros para além das suas credenciais do Epidemic Marketplace, o caminho para os ficheiros que representam o conteúdo digital e os metadados que a ele vão estar associados e ainda o nome da colecção a que estes vão estar agregados.

A Aplicação Cliente em Java tem a responsabilidade de construir um ficheiro RDF/XML com a sintaxe ORE, no qual está descrita a agregação e os recursos a ela agregados. Este ficheiro RDF/XML é então enviado por esta aplicação para o serviço web de Upload do Mediador, juntamente com as credenciais do Utilizador que despoletou a acção do serviço.

Mais uma vez o Mediador faz a verificação da autenticação do Utilizador junto do servidor LDAP. Após isto o Mediador utiliza a *Interface API-M – Fedora Management Service* para criar uma nova colecção no repositório, com o nome definido pelo Utilizador. De seguida, utilizando a mesma interface, cria os *Datastreams* correspondentes ao ficheiro e seus metadados e agrega ambos à colecção criada na interacção anterior.

Por fim o Mediador envia para o Utilizador o ficheiro RDF/XML com sintaxe ORE, como resposta à execução com sucesso do serviço.

3.3 API do Mediador

As tabelas (*Tabela 2*,

Tabela 4, *Tabela 5* e *Tabela 6*) apresentadas nesta secção, representam a API dos serviços web disponibilizados pelo Mediador do Epidemic Marketplace, especificando para cada um deles o URL base, os formatos suportados, o método HTTP utilizado, a necessidade de autenticação, o formato da resposta, os parâmetros da chamada ao serviço e exemplos de chamadas, caso aplicável.

No caso específico da *Tabela* 6 é feita a especificação da aplicação cliente construída na linguagem Java, para a funcionalidade de Upload do Mediador.

Tabela 2: API do Serviço Web de Pesquisa de Conteúdos

URL Base		https://epiwork.di.fc.ul.pt/mediator/search/objects/
Formatos Suportados		XML
Métodos HTTP Suportados		GET
Requer Autenticação		Sim (*HTTP Basic Authentication*)
Formato Resposta		XML
Parâmetros	[opcional] title	O nome dado ao recurso
	[opcional] creator	A entidade responsável pela criação do recurso
	[opcional] subject	O tema do recurso
	[opcional] description	Uma representação do recurso
	[opcional] publisher	A entidade responsável por tornar o recurso disponível
	[opcional] contributor	Uma entidade responsável por realizar contribuições para o recurso.
	[opcional] date	Uma data ou período de tempo associado a um evento no ciclo de vida do recurso
	[opcional] type	A natureza ou género do recurso
	[opcional] format	O formato do ficheiro ou as dimensões do recurso
	[opcional] identifier	Uma referência não ambígua do recurso num determinado contexto
	[opcional] source	Um recurso relacionado a partir do qual o recurso descrito deriva
	[opcional] language	A linguagem do recurso
	[opcional] relation	Um recurso relacionado
	[opcional] coverage	O tema temporal ou espacial do recurso.
	[opcional] rights	Informação dos direitos detidos no recurso e pelo recurso
Exemplos de chamadas ao Serviço de Pesquisa:		
https://epiwork.di.fc.ul.pt/mediator/search/objects/title/h1n1		
https://epiwork.di.fc.ul.pt/mediator/search/objects/creator/xldb		
https://epiwork.di.fc.ul.pt/mediator/search/objects/subject/health/language/en		

Tabela 3: API do Serviço Web de Pesquisa de Colecções

URL Base		https://epiwork.di.fc.ul.pt/mediator/search/collections/
Formatos Suportados		XML
Métodos HTTP Suportados		GET
Requer Autenticação		Sim (*HTTP Basic Authentication*)
Formato Resposta		XML
Parâmetros	[opcional] title	O nome dado ao recurso
	[opcional] creator	A entidade responsável pela criação do recurso
	[opcional] subject	O tema do recurso
	[opcional] description	Uma representação do recurso
	[opcional] publisher	A entidade responsável por tornar o recurso disponível
	[opcional] contributor	Uma entidade responsável por realizar contribuições para o recurso.
	[opcional] date	Uma data ou período de tempo associado a um evento no ciclo de vida do recurso
	[opcional] type	A natureza ou género do recurso
	[opcional] format	O formato do ficheiro ou as dimensões do recurso
	[opcional] identifier	Uma referência não ambígua do recurso num determinado contexto
	[opcional] source	Um recurso relacionado a partir do qual o recurso descrito deriva
	[opcional] language	A linguagem do recurso
	[opcional] relation	Um recurso relacionado
	[opcional] coverage	O tema temporal ou espacial do recurso.
	[opcional] rights	Informação dos direitos detidos no recurso e pelo recurso
Exemplos de chamadas ao Serviço de Pesquisa:		
https://epiwork.di.fc.ul.pt/mediator/search/collections/title/h1n1		
https://epiwork.di.fc.ul.pt/mediator/collections/search/creator/xldb		
https://epiwork.di.fc.ul.pt/mediator/search/collections/subject/health/language/en		

Tabela 4: API do Serviço Web de Download

URL Base	https://epiwork.di.fc.ul.pt/mediator/fetch/
Formatos Suportados	XML
Métodos HTTP Suportados	GET
Requer Autenticação	Sim (*HTTP Basic Authentication*)
Formato Resposta	XML (sintaxe OAI-PMH)
Parâmetros	
id	O ID do conteúdo que o Utilizador pretende obter
Exemplo de chamada ao Serviço de Download:	
https://epiwork.di.fc.ul.pt/mediator/fetch/id/18	

Tabela 5: API do Serviço Web de Upload

URL Base	https://epiwork.di.fc.ul.pt/mediator/upload
Formatos Suportados	RDF/XML (sintaxe OAI-ORE)
Métodos HTTP Suportados	POST
Requer Autenticação	Sim (*HTTP Basic Authentication*)
Formato Resposta	RDF/XML (sintaxe OAI-ORE)
Parâmetros	
oreFile	O ficheiro no formato RDF/XML e com a sintaxe OAI-ORE que contém a agregação e que é construído pela aplicação cliente Java, disponibilizada os utilizadores do Epidemic Marketplace

Tabela 6: Especificação da Utilização da Aplicação Cliente para a Funcionalidade de Upload

Linguagem		Java
Parâmetros	pasta	A localização onde foi extraído o ficheiro JAR
	classe	A classe Client.class resultante da extracção do JAR
	username	O nome de utilizador
	password	A palavra-passe do utilizador
	colecção	O nome da Colecção
	numRecursos	Um inteiro que representa o número de Recursos a introduzir no repositório
	caminhoRecurso	O *path* para o Recurso que se pretende introduzir no repositório
	caminhoMetadata	O path para o ficheiro Dublin Core correspondente ao Recurso que se pretende introduzir no repositório
Sintaxe		java <pasta>/<classe> <username> <password> <colecção> <numRecursos> [<caminhoRecurso1> ... <caminhoRecursoN>] [<caminhoMetadata1> ... <caminhoMetadataN>]
Exemplo de chamada à Aplicação Cliente para a Funcionalidade de Upload:		
java javaclient/Client hmferreira 12345 ColecçãoTeste 2 /home/hmferreira/ficheiro1 /home/hmferreira/ficheiro2 /home/hmferreira/metadata1 /home/hmferreira/metadata2		

Capítulo 4 - Implementação

4.1 Utilização de Servlets

As *servlets* são consideradas extensões dos servidores. São basicamente classes na linguagem de programação Java que estão preparadas para processar requisições e respostas de forma dinâmica. Utilizam normalmente o protocolo HTTP, o que se verifica também na implementação deste projecto.

Foram criadas 3 *Servlets* durante a implementação dos serviços de mediação, estando associada a cada um dos 3 principais serviços fornecidos pelo Mediador, uma *Servlet* diferente. Estas, como já ficou subentendido, são então responsáveis por tratar as requisições realizadas pelos utilizadores do Mediador, assim como pelo envio das respostas relativas às mesmas requisições.

É também através das *servlets* que é exigido aos utilizadores que, juntamente com cada pedido aos serviços web, enviem também as suas credenciais do Epidemic Marketplace através do método Autenticação de Acesso Básico HTTP (*Basic Access Authentication HTTP*). Caso não o façam verão negado o seu acesso aos mesmos serviços sendo que será retornada pela *servlet* que tratou o pedido, a mensagem de erro *"HTTP Status 401 – Unauthorized"*.

Dentro destas *Servlets* foi utilizada a *Java API for RESTful Web Services(JAX-RS)* [36]. Esta API na linguagem de programação Java permite a criação de serviços web que vão de encontro ao estilo arquitectural REST e utiliza anotações que visam simplificar o desenvolvimento e a implantação desses mesmos serviços web.

Algumas das anotações fornecidas por esta API e que fazem o mapeamento de uma classe Java num recurso web, são:

- ✓ **@Path**, especifica o caminho relativo para a classe de recursos.
- ✓ @**GET, @PUT, @POST, @DELETE**, especificam o tipo de pedido HTTP que é tratado por um determinado método da classe.
- ✓ **@Produces**, especifica os *mime-types* que são retornados por determinado método da classe.
- ✓ **@Consumes**, especifica os *mime-types* do pedido, que são aceites por determinado método da classe.

- ✓ **@PathParam, @QueryParam, @HeaderParam, @CookieParam, @MatrixParam, @FormParam,** especifica a origem dos valores recebidos como parâmetros por determinado método da classe.
  - Exemplos:
    - **@PathParam** significa que os parâmetros são extraídos do URL do pedido.
    - **@QueryParam** significa que os pârametros são extraídos do parâmetro de consulta do URL.
    - **@HeaderParam** significa que os parâmetros são extraídos dos cabeçalhos HTTP do pedido.
    - **@CookieParam** significa que os parâmetros são extraídos através da *Cookie* incluída no pedido HTTP.

4.2 Autenticação local dos utilizadores

O conjunto de serviços web implementados pelo Mediador do Epidemic Marketplace será somente disponibilizado aos utilizadores autorizados desta plataforma.

Para usufruírem destes serviços, no momento em que realizam os pedidos os utilizadores devem também fornecer as suas credenciais (nome de utilizador e palavra-chave).

Visto que se trata de um conjunto de serviços fornecidos através de uma interface Restful (em que não há aproveitamento do contexto entre transições de estado), cada solicitação a determinado serviço deve ser acompanhado pelas credenciais do utilizador.

Uma vez que as solicitações são realizadas através de pedidos HTTP (também este um protocolo *stateless*, ou seja, que não guarda a informação de uma prévia autenticação de determinado utilizador), o envio das credenciais deve ser feito através do método de Autenticação de Acesso Básico HTTP (*Basic Access Authentication HTTP*). Este método permite que um navegador web (browser) ou um programa cliente, forneçam credenciais aquando da concretização de um pedido HTTP, como demonstrado na *Figura 9*.

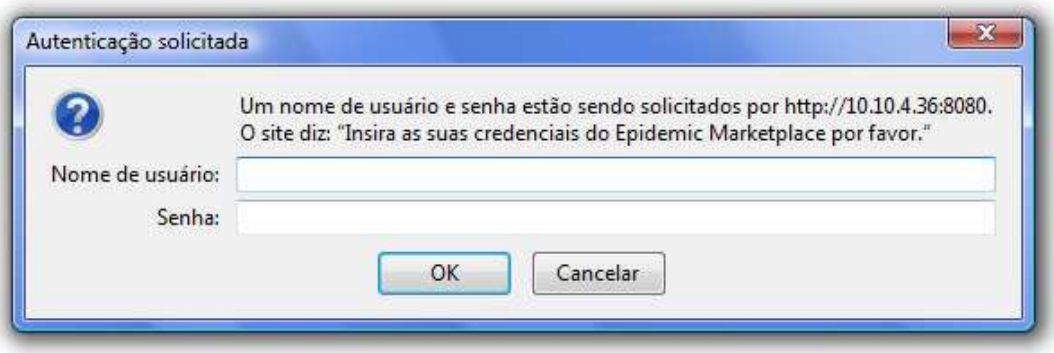


Figura 9: Requisição de Autenticação

A verificação das credenciais dos utilizadores é da responsabilidade da Classe *LoginBean.java*. A Classe *Servlet* responsável por receber o pedido envia para a classe *LoginBean.java* o objecto *HttpServletRequest* que contém a informação sobre o pedido HTTP recebido e sobre os cabeçalhos do mesmo, entre os quais está o cabeçalho *Authorization* que contém as credenciais para a autenticação via HTTP.

A classe *LoginBean.java* retira do cabeçalho *HTTP Authorization* a *String* que contém as credenciais, descodificando-as para poder utilizá-las junto do servidor OpenLDAP, no processo de autenticação do utilizador.

As propriedades relativas ao servidor OpenLDAP estão identificadas num ficheiro de propriedades com o nome "*PostgreSQL.properties*", junto do qual a classe obtém essas mesmas propriedades, que lhe vão permitir configurar a ligação ao servidor.

A classe utiliza a API do JNDI (Java Naming and Directory Interface) para realizar a ligação ao servidor OpenLDAP. O JNDI permite aos objectos e aplicações Java a utilização de um poderosa e transparente interface de acesso aos serviços de directório, como é o caso do LDAP.

Para se conectar ao servidor OpenLDAP, a classe *LoginBean.java* começa por obter a referência a um objecto que implemente a interface *DirContext*. Para tal, utiliza um objecto do tipo *InitialDirContext* que recebe como argumento um *Hashtable*. O objecto *Hashtable* passado como argumento contém várias entradas com informação,

entre as quais o “hostname”, porto e as classes do fornecedor de serviço JNDI a serem utilizadas.

Este procedimento é visível na Figura 10 que apresenta o excerto de código que faz a ligação ao servidor *OpenLDAP* através da interface JNDI.

```
public static DirContext createDirContext(String ldapURL) throws NamingException {
        Hashtable<String, String> env = new Hashtable<String, String>();
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, ldapURL);
        env.put(Context.SECURITY_AUTHENTICATION, securityAuth);
        env.put(Context.SECURITY_PRINCIPAL, ldapBindDN);
        env.put(Context.SECURITY_CREDENTIALS, password);
        return new InitialDirContext(env);
}
```

Figura 10: Ligação ao servidor OpenLDAP através da Interface JNDI

Após o estabelecimento da ligação, a classe *Login.java* verifica então as credenciais do utilizador junto do servidor, retornando um *boolean* que representa o sucesso ou insucesso da autenticação.

4.3 Serviço Web de Pesquisa de Conteúdos

A primeira funcionalidade implementada pelo Mediador do Epidemic Marketplace foi a possibilidade de os utilizadores realizarem a pesquisa de objectos digitais existentes no Repositório (sejam eles Conteúdos ou Colecções), através da consulta sobre os metadados associados aos mesmos.

A diferença entre a pesquisa de objectos digitais e a pesquisa de colecções, está somente no URL Base utilizado em cada um dos serviços. Para pesquisar Conteúdos Digitais o utilizador deve realizar um pedido HTTP GET ao seguinte URL Base:

https://epiwork.di.fc.ul.pt/mediator/search/objects/

Por outro lado, caso pretenda pesquisar colecções contidas no repositório, deve realizar um pedido HTTP GET ao seguinte URL Base:

https://epiwork.di.fc.ul.pt/mediator/search/collections/

Esta funcionalidade foi implementada pelo Mediador com recurso à Interface de Pesquisa Fedora (*Fedora Search Interface*). Esta disponibiliza um mecanismo de busca e navegação no repositório Fedora, através da consulta dos termos indexados aos metadados.

O *servlet* responsável por receber o pedido HTTP GET correspondente à chamada ao serviço de pesquisa é representado pela classe *SearchResource.java*. Através do objecto *HttpServletRequest* recebido como parâmetro pelo método *doGet()*, que trata a recepção do pedido, o *servlet* vai verificar junto do servidor LDAP a validade das credenciais fornecidas e obter a partir do URL enviado pelo utilizador, os parâmetros de pesquisa por este requeridos e os seus respectivos valores.

Estes dados são então reencaminhados para a classe *SearchBean.java*, onde ocorre todo o processamento necessário à execução da pesquisa e formulação da resposta. Esta classe constrói um pedido à Interface de Pesquisa Fedora, no qual inclui os critérios de pesquisa identificados pelo Utilizador. A resposta da Interface surge no formato XML e nela vêm identificados todos os IDs, que satisfazem a consulta. Para cada um dos IDs presentes na resposta da Interface Fedora, a classe *SearchBean.java* vai realizar um pedido OAI-PMH utilizando o verbo *getRecord* de modo a obter o registo de metadados Dublin Core do conteúdo. É feita a compilação de todos os registos de metadados de todos os conteúdos resultantes da pesquisa, num único ficheiro XML com a sintaxe OAI-PMH, sendo o conteúdo do mesmo a resposta a ser enviada para o utilizador.

Para fazer a distinção entre os objectos digitais do repositório que representam Conteúdos e os que representam Colecções, é feita uma chamada à interface Fedora para se obter a listagem dos Datastreams do objecto. Caso o objecto não possua nenhum *"Thumbnail Datastream"*, é considerado uma Colecção.

A construção dos URLs com os pedidos ao serviço de pesquisa do Mediador, devem seguir a sintaxe dos URLs apresentados na API do mesmo.

Para além dos URL Base do serviço, devem ser adicionados os parâmetros e respectivos valores, de forma alternada e separados pelo caracter "/". Caso não seja adicionado nenhum parâmetro e respectivo valor a seguir ao URL Base, o pedido é interpretado como se fosse uma pesquisa de todos os Conteúdos ou Colecções existentes no repositório.

- **Sintaxe:**
  https://epiwork.di.fc.ul.pt/mediator/search/objects/param1/value1/param2/value2...

Exemplos:

- ✓ **Pesquisa sem parâmetros:**
  - https://epiwork.di.fc.ul.pt/mediator/search/objects/ - Retorna os registos de metadados equivalentes a uma pesquisa de todos os Conteúdos existentes no repositório.
  - https://epiwork.di.fc.ul.pt/mediator/search/collections/ - Retorna os registos de metadados equivalentes a uma pesquisa de todas as Colecções existentes no repositório.

- ✓ **Pesquisa simples:**
  - https://epiwork.di.fc.ul.pt/mediator/search/collections/title/H1N1 - Retorna os registos de metadados para as Colecções que contenham a *String* "H1N1" no campo *title*.
  - https://epiwork.di.fc.ul.pt/mediator/search/objects/creator/Epiwork - Retorna os registos de metadados para os Objectos que contenham a *String* "Epiwork" no campo *creator*.

- ✓ **Pesquisa composta:**
  - https://epiwork.di.fc.ul.pt/mediator/search/collections/title/H1N1/creator/Epiwork/ - Retorna os registos de metadados para as Colecções que contenham a *String* "H1N1" no campo *title* e que em simultâneo contenham a *String* "Epiwork" no campo *creator*.

- https://epiwork.di.fc.ul.pt/mediator/search/objects/subject/health/date/2009-05-29 - Retorna os registos de metadados para os Conteúdos, existentes no repositório, que contenham a *String* "health" no campo *subject* e que em simultâneo contenha a data "2009-05-29" no campo *date*.

É necessário notar que os termos pesquisados não são avaliados em função de letras maiúsculas ou minúsculas, isto é, são *case insensitive*.

Foi tomado o cuidado de codificar os valores utilizados na pesquisa em UTF-8, de modo a não surgirem problemas com caracteres especiais ou acentuados.

```
<OAI-PMH xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/ http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
 −<GetRecord>
   −<record>
     −<header>
        <identifier>oai:example.org:changeme:47</identifier>
        <datestamp>2009-11-14T15:27:04Z</datestamp>
        <setSpec>objects</setSpec>
       +<downLink></downLink>
       +<recordLink></recordLink>
      </header>
     −<metadata>
       −<oai_dc:dc xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/ http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
         +<dc:title></dc:title>
          <dc:creator>LASIGE </dc:creator>
          <dc:subject>Twitter message dataset </dc:subject>
         +<dc:description></dc:description>
          <dc:publisher>Epiwork </dc:publisher>
          <dc:contributor>Patricia Sousa </dc:contributor>
          <dc:date>2009-11-14 </dc:date>
          <dc:type>dataset </dc:type>
          <dc:format>Text/tab-separated-values </dc:format>
          <dc:identifier>dataset-twitter-003 </dc:identifier>
          <dc:source>http://epiwork.di.fc.ul.pt/collector </dc:source>
          <dc:language>English </dc:language>
         +<dc:relation></dc:relation>
         +<dc:coverage></dc:coverage>
         +<dc:rights></dc:rights>
        </oai_dc:dc>
      </metadata>
    </record>
  </GetRecord>
</OAI-PMH>
```

Figura 11: Exemplo do XML de resposta do serviço de pesquisa

Na *Figura 11* é possível um exemplo de um XML de resposta a um pedido realizado ao serviço web de pesquisa do Mediador. O XML respeita a sintaxe do protocolo OAI-PMH. Neste caso específico a pesquisa é feita sobre um Conteúdo e como tal, foram adicionadas duas *subtags* à *tag* <*header*>.

- A *tag* <*downLink*> contém o URL do pedido ao serviço web de download do Mediador, que permite ao utilizador obter este Conteúdo em particular.
- A *tag* <*recordLink*> fornece ao utilizador o URL para o registo de metadados individual do conteúdo.

A *tag* <*metadata*> contém os metadados Dublin Core associados ao Conteúdo.

4.4 Serviço Web de Download de Conteúdos

A segunda funcionalidade implementada pelo Mediador do Epidemic Marketplace introduz a possibilidade de os utilizadores realizarem o download de Conteúdos digitais.

Os utilizadores autorizados do Epidemic Marketplace podem assim aceder e descarregar os Conteúdos existentes no repositório, bastando-lhes para tal realizar um pedido HTTP GET ao URL Base (identificado na API) do serviço de download de conteúdos. Ao URL Base do pedido deve ser acrescentado o parâmetro "*id*" e o seu respectivo valor, que representa o identificador único no repositório, do Conteúdo que o utilizador pretende descarregar.

- URL Base: https://epiwork.di.fc.ul.pt/mediator/fetch/
- Sintaxe: https://epiwork.di.fc.ul.pt/mediator/fetch/id/[idDoConteudo]

Exemplo de um pedido ao serviço web de download de conteúdos do Mediador, para descarregamento do Conteúdo com o ID 18:

- https://epiwork.di.fc.ul.pt/mediator/fetch/id/18

O *servlet* responsável por receber o pedido HTTP GET correspondente à chamada ao serviço de download é representado pela classe *FetchResource.java*. Através do objecto *HttpServletRequest* recebido como parâmetro pelo método *doGet()* que trata a recepção do pedido, o *servlet* vai verificar a validade das credenciais fornecidas e obter a partir do URL enviado pelo utilizador, o parâmetro *id* e o seu respectivo valor.

Estes dados são então reencaminhados para a classe *FetchBean.java*, onde ocorre todo o processamento necessário à execução da pesquisa e formulação da resposta. Esta classe constrói um pedido à Interface de Pesquisa Fedora, no qual inclui o ID do conteúdo nos critérios de pesquisa. A resposta da Interface surge no formato XML e nela vêm identificadas as informações sobre o Conteúdo pretendido pelo utilizador (título do conteúdo, criador do conteúdo, data de criação, etc).

Da resposta recebida, a classe FetchBean.java vai retirar o título guardando-o numa variável do tipo *String*. Esta operação é necessária, uma vez que ao se descarregar um objecto digital armazenado no repositório através da *API-A Lite – Fedora Acess Service*, este vai ficar armazenado no sistema de ficheiros local com o ID que possuía no repositório como nome em detrimento do seu nome original.

A classe FetchBean.java constrói um novo pedido, desta vez à *API-A Lite – Fedora Acess Service* para obter o Conteúdo e guardá-lo no seu sistema de ficheiros local. Uma vez que este trabalho visa construir serviços web com base numa interface RESTful e segundo esta metodologia, é aconselhada uma normalização da estrutura das respostas enviadas para os clientes dos serviços, optou-se por não se enviar os Conteúdos obtidos do repositório no seu formato binário. À semelhança dos outros dois restantes serviços de mediação, a resposta ao serviço de download é entregue no formato XML e como tal, foi necessário codificar os ficheiros do seu formato binário para Base64, de modo a incorporá-los no ficheiro XML de resposta a ser enviado para o Utilizador.

O Conteúdo binário é codificado em Base64 através do recurso à classe *BASE64Encoder* fornecida pela Sun, como é visível no seguinte excerto de código:

```
byte[] array64 = new sun.misc.BASE64Encoder().encode(binArray).getBytes();
```

Posteriormente, é construído um ficheiro XML através da geração de um novo objecto da classe *Document* fornecido pela W3Schools. A criação desde *Document* permite a manipulação de um conjunto de dados como se de um XML se tratasse:

```
DocumentBuilderFactory fact = DocumentBuilderFactory.newInstance();
DocumentBuilder parser = fact.newDocumentBuilder();
Document doc = parser.newDocument();
```

A este objecto é então adicionado a codificação do Conteúdo em Base64 e o título do conteúdo, sendo o objecto finalmente convertido num ficheiro XML e enviado para o Utilizador que fez a requisição do serviço.

4.5 Serviço Web de Upload de Conteúdos

A implementação do serviço web de Upload de conteúdos foi realizada com base na proposta do projecto OAI-ORE (Open Archives Initiative – Object Reuse and Exchange) [29], o qual define uma norma para a identificação, descrição e permuta de objectos digitais compostos. É este o nome dado aos agregados de recursos informáticos que permitem a combinação de diferentes tipos/formatos de conteúdos numa mesma agregação.

Uma vez que o Mediador pretende disponibilizar serviços que facilitem a interacção dos utilizadores com o repositório do Epidemic Marketplace e o acesso aos conteúdos nele armazenados e em simultâneo pretende preservar os conceitos da metodologia REST e as normas da abordagem OAI-ORE, foi necessário uma solução um pouco mais rebuscada para a construção do serviço web de Upload.

A abordagem OAI-ORE introduz o conceito de Mapa de Recursos que é representado por um grafo RDF (Resource Description Framework), o qual descreve uma agregação, o agregado de recursos que a compõem e as relações entre os mesmos. O grafo RDF que representa o Mapa de Recursos pode, no entanto, ser representado em qualquer serialização da sintaxe RDF, assim como em outros formatos entre os quais

surgem o Atom XML, N3 ou Turtle. Como tal, o projecto OAI-ORE propõe a utilização da sintaxe RDF/XML na serialização destes mesmos Mapas.

A união deste conjunto de critérios e condições leva a que a realização do Upload de um conteúdo para o repositório do Epidemic Marketplace, por parte de um utilizador, consista no envio de um ficheiro com o formato RDF/XML e com uma sintaxe que respeite as normas definidas pela abordagem OAI-ORE. Este ficheiro seria enviado pelo utilizador para o Mediador, que posteriormente o disponibilizaria no repositório.

Este ficheiro representa a agregação e nele deve constar a descrição da agregação, dos recursos que esta agrega, das relações entre os recursos e dos metadados, segundo o padrão Dublin Core, associados aos respectivos recursos. Na solução escolhida para a implementação desta funcionalidade foi ainda definido que os recursos a serem armazenados no repositório, deveriam ser transmitidos através da sua inclusão no ficheiro RDF/XML enviado para o Mediador por parte dos utilizadores.

No entanto e uma vez que os serviços web disponibilizados pelo Mediador do Epidemic Marketplace pretendem ser intuitivos, fáceis de utilizar e uma forma de motivar os utilizadores a interagirem com o Repositório, delegar no utilizador a responsabilidade de construir o ficheiro RDF/XML acima descrito, seria uma opção que não podia ser considerada, visto que ia no sentido contrário ao dos objectivos do Mediador, tendo em conta a complexidade inerente à realização deste processo.

Posteriormente, é ainda necessário a construção de um pedido HTTP POST, no qual o utilizador envia o ficheiro com o formato RDF/XML, para o Mediador. A construção deste pedido no qual é encapsulado o ficheiro a enviar, não é também uma questão completamente trivial, uma vez que requer alguns conhecimentos ao nível de programação.

Como tal, a solução encontrada para a implementação do serviço web de Upload de conteúdos foi a construção de uma biblioteca em Java, a ser disponibilizada aos utilizadores do Epidemic Marketplace, a qual se responsabilizará pela criação do ficheiro RDF/XML assim como pelo envio deste para o Mediador através de um pedido HTTP POST.

A informação em maior detalhe relativa ao funcionamento desta biblioteca, poderá ser encontrada num documento anexo a este (ANEXO I - *Client Application to the Upload Functionality in the Epidemic Marketplace Mediator*).

A biblioteca em Java é então responsável por todo o processamento do lado do cliente e pela chamada ao serviço web de upload do Mediador, através do método HTTP POST no qual envia não só as credenciais do utilizador como o ficheiro no formato RDF/XML e com a sintaxe definida pela abordagem OAI-ORE, no qual consta a definição da Agregação e dos seus Recursos Agregados.

O pedido HTTP POST é enviado para o URL Base do serviço, que neste caso é também o URL final, uma vez que não há a adição de parâmetros ao mesmo:

- URL Base: https://epiwork.di.fc.ul.pt/mediator/upload/

Do lado do servidor (Mediador), o *servlet* responsável por receber o pedido HTTP POST correspondente à chamada ao serviço de upload é representado pela classe *UploadResource.java*. O objecto *HttpServletRequest* recebido como parâmetro pelo método *doPost()* que trata a recepção do pedido é utilizado para verificar a validade das credencias do utilizador junto do servidor LDAP e é de seguida reencaminhado para a classe *UploadBean.java*, onde ocorre todo o processamento necessário à execução da inserção dos conteúdos no repositório e à construção da resposta a ser transmitida ao utilizador

Nesta classe é recebido o objecto *HttpServletRequest* através do qual é obtido o ficheiro RDF/XML. É posteriormente realizado o *parsing* do ficheiro XML.

Parsing é processo de representação de um ficheiro XML num objecto DOM (*Document Object Model*) *Document*. Este objecto permite a manipulação de uma estrutura de dados em forma de árvore, através de uma API que permite um conjunto de operações como a inserção, remoção e actualização de nós, entre outras.

Feito o *parsing* são obtidas do ficheiro as informações necessárias ao processamento desta classe. É obtido o nome da Colecção representada pela Agregação e verificado junto da Interface de Pesquisa Fedora, se já existe no repositório alguma Colecção com este nome. Caso ainda não exista, será criado um novo objecto digital do

tipo Colecção através da *API-M – Fedora Management Service*, caso já exista a inserção dos novos Conteúdos será feita dentro da mesma.

É também obtida a codificação em Base64, dos conteúdos a serem colocados no repositório, que se encontrava encapsulada no ficheiro XML. Os conteúdos são transformados em ficheiros binários a partir da descodificação do texto em Base64.

São verificados se os Recursos Agregados correspondem a ficheiros ou a URLs, uma vez que existe a possibilidade de mapeamento de dados no repositório através de ficheiros físicos armazenados localmente ou através de URLs para a localização remota dos mesmos.

Os metadados correspondentes a cada um dos conteúdos são também extraídos do XML obtido pela classe *UploadBean.java,* para que possam ser posteriormente introduzidos no repositório, sob a forma de *Datastreams* indexados aos Conteúdos que descrevem.

Após a criação da Colecção (caso esta ainda não existisse) e do upload de todos os Conteúdos e correspondentes metadados para o repositório do Epidemic Marketplace, é eliminado do ficheiro XML a codificação Base64 dos conteúdos agregados, ficando este com um aspecto mais limpo e legível e sendo posteriormente enviado para utilizador, na interacção que simboliza a resposta do serviço e o fim do seu processamento. Este ficheiro XML enviado na resposta para o Utilizador, permite-lhe ter a visualização da estrutura que acabou de criar dentro do repositório. Sendo que o ficheiro representa um Mapa de Recursos, no qual está especificado uma Agregação (ou Colecção no contexto do repositório) e os seus Recursos Agregados (aos quais correspondem os *Datastreams* e correspondentes metadados introduzidos no repositório).

4.6 Decisões de Implementação

Ao longo da realização do projecto foram tomadas várias decisões de implementação que em algumas situações levaram o trabalho num sentido diferente daquele que estava inicialmente pensado e planeado. Estas decisões e pontuais

alterações de planos prendem-se com mudanças de requisitos, constrangimentos a nível de tecnologia disponível e de conhecimentos, idealização de uma melhor solução do que a inicialmente calculada, etc.

Nesta secção pretende-se explicar quais as decisões de implementação que foram necessárias tomar durante a realização do projecto, o motivo que despoletou a tomada dessas decisões e o modo como estas interferiram ou não no desenrolar do trabalho.

- **Criação de uma página web para os serviços de mediação.**

  Durante o planeamento deste trabalho, foi equacionada a criação de uma página web, de modo a fomentar a utilização dos serviços de mediação, não apenas por via de aplicações, mas também por utilizadores humanos.

  Como tal, durante os primeiros meses, foram desenvolvidas páginas web, com base na tecnologia JSP – *Java Server Pages* [37], (tecnologia que foi estudada e praticada pelo aluno durante esses meses, através da leitura de livros e realização de tutoriais) onde estavam representados os serviços do Mediador. Com o decorrer do projecto e uma vez que existia já um *frontend* para o repositório Fedora a ser utilizado pela plataforma do Epidemic Marketplace, decidiu-se que construir uma página para os serviços do Mediador não seria mais do que refazer algo que já existia. A partir desse momento o foco foi a interface RESTful com uma API para os serviços, tendo sido abandonado o desenvolvimento de páginas web através de JSP.

  Na *Figura 12* é possível observar a página inicial da Página Web do Mediador, na qual é pedido aos utilizadores que introduzam as suas credenciais do Epidemic Marketplace.

Figura 12: Página web do Mediador - Login

- **Utilização de JavaBeans.**

A decisão da utilização de JavaBeans está bastante ligada ao ponto anterior. JavaBeans é um modelo que descreve uma arquitectura de componentes para Java baseada em classes nessa Java. Estes componentes possuem uma ligação bastante fácil e intuitiva com páginas desenvolvidas em JSP, tornando-se bastante eficiente a ligação entre os dois e a troca de informação e dados entre as classes Java e as páginas web.

Entre outras características importantes, estes componentes permitiam por exemplo a persistência dos objectos, através de um serviço de serialização, ou seja, era possível a gravação do seu estado e a recuperação deste numa posterior utilização dos mesmos. Esta característica vai contra a ideologia REST que se pretendia ver

implementada no Mediador e na qual se indica que o contexto da comunicação não deve ser aproveitado entre transições de estado.

Este conflito de interesses lançou dúvidas quanto à opção de se utilizarem JavaBeans. Mais tarde, com a decisão de se abandonar a tecnologia JSP e a criação de páginas web, vem também a decisão de finalmente abandonar a implementação de JavaBeans.

- **Escolha do NetBeans como IDE.**

  A escolha do software NetBeans [38] como IDE para a realização deste trabalho acabou por ser natural e previsível uma vez que para além de se ter decidido trabalhar com JavaBeans, também estava prevista a utilização de JSPs. Dando o NetBeans a possibilidade de executar um servidor Tomcat interno ao IDE que permite a emulação das páginas JSP, a sua escolha foi imediata.

  Ao contrário dos dois pontos anteriores, o IDE NetBeans não deixou de ser utilizado, tendo o aluno trabalhado com este IDE durante toda a duração do projecto.

- **Realização de uma interface RESTful em detrimento de SOAP.**

  Serviços web são componentes que permitem que duas aplicações troquem informações entre si utilizando um canal de comunicação, que na maioria das vezes se traduz no protocolo HTTP. Os serviços web mais conhecidos e utilizados hoje em dia são WS-* (WSDL/SOAP) e REST.

  Uma das mais acesas discussões actualmente no que diz respeito à concretização de serviços web, baseia-se na escolha de qual destes dois padrões utilizar, quais as vantagens e desvantagens de cada um, quais as suas forças e limitações. Defensores de ambos os lados têm alimentado discussões que chegam a roçar o fanatismo.

  Pelo estudo que foi efectuado para a realização deste trabalho, na opinião do aluno não há um padrão que seja sempre superior ao outro ou que deva ser utilizado independentemente de tudo o resto. Cada um deles tem as suas limitações e virtudes, cabendo à equipa de desenvolvimento

dos serviços web perceber quais as características dos serviços que pretende fornecer e analisar qual destes dois padrões melhor se adequa aos requisitos.

No caso específico do Mediador, a opção recaiu sobre o estilo de arquitectura REST e nesta secção pretende-se justificar o porquê.

A principal vantagem da ideologia REST é sem dúvida a sua simplicidade. E a simplicidade é um dos principais requisitos não funcionais identificados na construção do Mediador. A utilização de serviços REST acaba por ser bastante semelhante à utilização do protocolo HTTP. Fazendo a analogia, submeter um formulário HTML através do método POST do protocolo, é no fundo o mesmo que utilizar REST. A diferença está no formato da resposta, que virá normalmente em HTML enquanto que no REST é exigida a devolução de uma estrutura mais normalizada na resposta, como por exemplo XML ou JSON.

Em comparação com o padrão SOAP, este possui uma série de especificações que acabam por tornar a sua utilização bastante complexa e demasiado burocrática. Para fundamentar esta ideia basta referir que existem actualmente mais de 15 especificações para o padrão SOAP e tomar conhecimento de todas elas e do seu funcionamento, acaba por se tornar humanamente impossível.

Os serviços REST caracterizam-se por uma maior performance em comparação com SOAP, o que vai de encontro a outro dos requisitos para os serviços web de Mediador: carga reduzida a ser transmitida no protocolo HTTP.

No que respeita ao SOAP têm surgido várias queixas devido ao *overhead* que a troca de mensagens neste padrão pode causar, uma vez que todas as mensagens precisam de ser encapsuladas dentro de um envelope SOAP, que não é mais do que um XML que segue uma normalização. Apenas uma pequena parte do conteúdo deste envelope é utilizado para a concretização do serviço e como tal, pode-se considerar que o resto do conteúdo do envelope é "lixo".

Em SOAP é requerida a existência de um contracto que faça a definição dos serviços fornecidos, ao qual se dá o nome de WSDL - *Web Services Description Language*. Esta característica pode ser uma mais valia num ambiente em que existe um conjunto relativamente grande de serviços, desenvolvidos por diferentes fornecedores em diferentes plataformas, para serem utilizados por diferentes aplicações. Neste tipo de casos é de todo conveniente que existe um contracto formal que defina os serviços. No caso do Mediador isto não acontece uma vez que a complexidade do ambiente que o envolve é bem mais reduzida, logo faz todo o sentido utilizar o padrão REST que é considerado substancialmente mais simples e objectivo.

Outro dos requisitos não funcionais identificados na idealização do Mediador está relacionado com a legibilidade dos serviços disponibilidades e do modo como são acedidos. O padrão REST vai de encontro a esta expectativa uma vez que os serviços REST são identificados por URIs, tornando-se mais compreensíveis ao olho humano.

- **Definição do esquema de autenticação**

  Sendo a plataforma Epidemic Marketplace essencialmente constituída por um repositório digital onde são armazenados dados epidemiológicos, médicos e sociais, entre outros, que implicam restrições de acesso e privacidade aos mesmos, é essencial que exista um esquema de autenticação que possa ser utilizado para assegurar o acesso autorizado aos serviços web fornecidos pelo Mediador.

  Como tal, era um dos requisitos do Mediador desde o início, que fosse implementado este esquema de autenticação. No entanto, era necessário notar que os serviços estavam a ser disponibilizados através de uma interface RESTful o que só por si já implicava algum cuidado na implementação deste esquema. Uma das limitações do REST está precisamente relacionada com a questão da segurança, pois ao contrário

de SOAP, não existem padrões de suporte para questões de segurança no estilo de arquitectura REST.

Uma vez que o REST se caracteriza por comunicações *stateless*, ou seja, em que não há aproveitamento nem preservação do estado entre as transições. Para ir de encontro a esta restrição, é necessário que todas as chamadas aos serviços web disponibilizados pelo Mediador sejam acompanhadas pelas credenciais dos Utilizadores que as concretizem.

Esta foi uma das questões que levou mais tempo a implementar e que implicou o maior número de artigos estudados. Foram consideradas várias opções, analisadas as vantagens e desvantagens para cada uma delas. De entre as várias possibilidades de concretização do esquema de autenticação avaliadas, a opção recaiu sobre o método de Autenticação de Acesso Básico HTTP (*Basic Access Authentication HTTP*).

Através deste método, as credenciais do utilizador são incorporadas no pedido HTTP de chamada ao serviço web, ficando armazenadas no cabeçalho *HTTP Authorization* com codificação em Base64. Este método é utilizado para cada chamada aos serviços web.

A escolha deste método deve-se a um conjunto de motivos. Alguns deles estão relacionados com a sua eficiência e simplicidade, e com o facto de ser suportado por todos os *web browsers* mais populares e utilizados.

No entanto, a utilização deste esquema de autenticação por si só não confere segurança ao sistema, uma vez que as credenciais dos utilizadores são passadas como texto em claro, no cabeçalho HTTP do pedido. Como tal foi combinada a utilização deste esquema em conjunto com o protocolo SSL (*Secure Socket Layer*).

É garantido assim que a comunicação é sempre transmitida através do protocolo HTTPS (*HyperText Transfer Protocol Secure*). Desta forma os dados são sempre transmitidos através de uma conexão criptografada, sendo verificada a autenticidade do cliente e do servidor através de certificados digitais.

O servidor onde está a correr o Mediador do Epidemic Marketplace possui um certificado emitido por uma Autoridade Certificadora (*Certification Authority*) confiável, garantindo a autenticidade do mesmo.

- **Utilização do software Fedora Repository relativo à versão de 2.2.2**

  Ainda antes de ser iniciado o projecto da construção do Mediador para o Epidemic Marketplace, já esta plataforma estava em funcionamento, já existindo inclusive um repositório e um GUI (Graphical User Interface) ou *frontend* para o mesmo repositório. A versão do Fedora Repository utilizada era então a 2.2.2 e o *frontend* era apenas compatível com esta versão. A utilização desta versão do Fedora Repository acabou por de algum modo ser uma imposição.

  Actualmente o Fedora Repository encontra-se na versão 3.4. No entanto, as alterações não têm sido muito profundas entre versões, sendo por isso possível afirmar que as alterações a realizar ao Mediador para que se compatibilize com a última versão do Fedora, não serão muito difíceis de concretizar.

- **Comunicação entre Mediador e LDAP através de HTTPS**

  Na fase inicial do projecto, numa altura em que este estava a ser desenvolvido num servidor de testes, pensou-se em criptografar a comunicação entre o Mediador e o LDAP, para efeitos de uma verificação segura das credenciais dos utilizadores.

  Para tal foi criado um certificado auto-assinado (*self signed*) para o servidor, de modo a possibilitar a comunicação HTTPS entre este e o protocolo LDAPS (*Lightweight Directory Access Protocol Secure*). Uma vez que o certificado era auto-assinado e não era fornecido por uma Autoridade Certificadora confiável, foi preciso recorrer a uma série de tutoriais e testes de modo a garantir junto do LDAP que os pedidos emitidos pelo Mediador, provinham de um servidor confiável.

A resolução deste problema, que está descrita em maior detalhe no *ANEXO* II – Ligação HTTPS entre Mediador e LDAP, levou bastante tempo e implicou o estudo de bastantes tutoriais e a realização de uma infinidade de testes.

O tempo gasto nesta solução acabou por se revelar infrutífero, uma vez que a necessidade de conexões criptografadas entre o Mediador e o LDAP foi dispensada, a partir do momento em que se decidiu que estes dois componentes iriam ficar alojados na mesma máquina.

E mesmo que houvesse a necessidade deste tipo de conexões, teria bastado a obtenção de um certificado digital através de uma Autoridade Certificadora confiável para assegurar a autenticidade do servidor, evitando assim todas estas operações desnecessárias.

- **Protocolo OAI-PMH**

  À semelhança da utilização da versão 2.2.2 do Fedora Repository, também a integração do protocolo OAI-PMH (*Open Archives Initiative - Protocol for Metadata Harvesting)*, acabou por ser de certo modo um requisito do Mediador, que já tinha sido considerado antes do projecto começar na forma de uma Tese de Mestrado.

  Inicialmente estava prevista a implementação de raíz deste protocolo que determina um padrão baseado na partilha e divulgação de metadados entre repositórios.

  De notar que todos os objectos digitais contidos no repositório Fedora, contêm à partida um registo primário de metadados Dublin Core, mesmo que no momento da sua criação não tenha sido fornecido nenhum registo de metadados a ser indexado ao objecto.

  No entanto, após um estudo mais completo sobre a versão do Fedora Repository utilizada, foi possível perceber que esta implementava uma *Fedora OAI Provider Interface* para a utilização deste protocolo. Esta interface está de acordo a sintaxe padrão do protocolo OAI-PMH e processa todas as solicitações OAI-PMH (v2.0) válidas.

Como tal, ao invés de realizar a implementação de raíz deste protocolo como inicialmente idealizado, o Mediador concretiza a integração do mesmo através da realização de solicitações a esta interface, utilizando os verbos definidos para o efeito.

- **Abordagem OAI-ORE**

  Também a abordagem OAI-ORE (*Open Archives Initiative - Object Reuse and Exchange*) pode ser considerada como um pré-requisito do projecto, uma vez que a sua utilização já se encontrava planeada ainda antes do arranque da construção do Mediador do Epidemic Marketplace.

  Esta abordagem fornece uma *framework* para a gestão de agregações de objectos digitais uniformizando as relações entre estes.

  Os repositórios podem ser vistos como interfaces de armazenamento remoto, onde os conteúdos são representados e acedidos através de um Mapa de Recursos OAI-ORE.

  Neste Mapa de Recursos e segundo a sintaxe ORE, o termo aplicado a um conjunto de recursos é Agregação, sendo que esta descreve recursos aos quais se aplica o termo Recursos Agregados. De forma a representar uma estrutura hierárquica, as Agregações podem estar elas próprias contidas noutras Agregações.

  Uma Agregação pode ser considerado um conceito abstracto, no entanto, a serialização do Mapa de Recursos (no formato RDF/XML, por exemplo) fornece uma representação concreta da mesma.

  Concentrar toda a estrutura de um repositório traduzida num Mapa de Recursos pode revelar-se de grande utilidade em casos de migração de dados entre repositórios diferentes. O Mapa de Recursos facilitaria bastante esta tarefa, sendo apenas necessário replicar a sua estrutura no novo repositório.

  No entanto, e uma vez que neste projecto o foco principal é a construção de serviços de mediação de acesso aos dados armazenados no repositório digital, a utilização do Mapa de Recursos da forma acima

descrita não seria uma mais valia para este trabalho, pois não faz parte dos objectivos do mesmo a construção de uma solução que facilite o processo de migração dos dados de um repositório Fedora, para um novo repositório.

Revelou-se por isso necessário encontrar uma forma de introduzir a abordagem OAI-ORE no contexto dos serviços web do Mediador, de forma a que se tornasse uma mais valia a sua integração.

A solução encontrada passa por utilizar a abordagem OAI-ORE e o conceito de Mapa de Recursos na funcionalidade de upload do Mediador, de forma a aumentar as potencialidades da mesma. Deste modo, os utilizadores podem realizar o upload de vários conteúdos, de diferentes formatos (o que vai de encontro ao conceito de agregação de objectos digitais compostos), numa só chamada ao serviço.

Para além disso, a cada recurso pode ser associado um ficheiro de metadados Dublin Core, sendo esta associação preservada no momento de armazenamento dos conteúdos no repositório.

Os Utilizadores apenas necessitam de definir o nome da Agregação (que será uma Colecção no repositório) e quais os conteúdos e correspondentes metadados que serão os Recursos Agregados (transformados em Conteúdos no repositório), pertencentes a essa mesma Agregação.

- **Criação de uma Aplicação Cliente em Java, para a funcionalidade de Upload do Mediador**

  Ao contrário dos serviços web de pesquisa e de download fornecidos pelo Mediador do Epidemic Marketplace, o serviço de upload tem um nível de exigência relativamente maior do lado do cliente, para que este possa concretizar o pedido.

  Enquanto que os primeiros são acedidos através do método HTTP GET e a única informação que é necessária enviar, para além do URL do pedido com os parâmetros correctos, diz respeito às credenciais do utilizador.

Para além de ser necessário a concretização de um pedido HTTP POST para realizar a chamada ao serviço, é preciso também garantir que no pedido seguem não só as credenciais (como já acontecia nos outros serviços), como também o ficheiro no formato RDF/XML com a sintaxe definida pela abordagem OAI-ORE, onde se encontra o Mapa de Recursos que representa a Agregação de Recursos a serem introduzidos no repositório.

Uma vez que os serviços web disponibilizados pelo Mediador do Epidemic Marketplace pretendem ser intuitivos, fáceis de utilizar e uma forma de motivar os utilizadores a interagirem com o Repositório, delegar no utilizador a responsabilidade de executar todas estas operações com um grau de complexidade considerável, iria no sentido contrário destes objectivos. Deste modo foi pensada uma solução um pouco mais rebuscada, de modo a manter as premissas de facilitar a tarefa do utilizador e simplificar a sua interacção com o Mediador.

A solução passou por criar uma aplicação em Java que será disponibilizada aos utilizadores registados do Epidemic Marketplace, que basicamente faz o trabalho todo pelo Utilizador. Transforma os conteúdos em Base64, constrói um Mapa de Recursos que descreve uma Agregação num ficheiro RDF/XML com a sintaxe ORE e encapsula no mesmo os recursos e respectivos metadados fornecidos pelo utilizador. Por fim, constrói um pedido http POST que chama o serviço de upload do Mediador, enviando as credenciais do utilizador e o ficheiro RDF/XML.

- **Criação de um esquema de controlo de acesso baseado num sistema de permissões**

Ao se registarem no Epidemic Marketplace os utilizadores lhes ser atribuído um *role*, que os identifica como pertencendo a uma instituição ou categoria de utilizadores. Para cada *role* existem permissões e direitos de acesso diferentes. Todos os objectos digitais do

repositório estão associados ao utilizador que o criou e consequentemente, estão associados às permissões definidas para o *role* a que o mesmo pertence.

Foi inicialmente pensada para o Mediador, a necessidade de um esquema de controlo de acesso e de verificação de permissões criado de raiz. No entanto, os estudo do Fedora Repository levou à conclusão de que este já possui um módulo de execução de políticas de controlo de acesso baseado em XACML (*eXtensible Access Control Markup Language*), que consiste numa linguagem declarativa de políticas de controlo de acesso implementada em XML e num modelo de processamento que descreve a forma de interpretar as políticas. Ao realizar a comunicação com as interfaces disponibilizadas pelo Fedora Repository, o Mediador envia também as credenciais do utilizador que despoletou a acção do serviço (garantindo desde logo que aquele utilizador está autenticado no sistema, uma vez que passou pelo esquema de autenticação do serviço).

Por sua vez, o sistema Fedora verifica o *role* correspondente às credenciais de utilizador recebidas e através do módulo baseado em XACML, define qual o nível de acesso para o mesmo.

Capítulo 5 - Testes de Desempenho e de Escalabilidade

À semelhança qualquer outro projecto de desenvolvimento de software, também o desenvolvimento do Mediador do Epidemic Marketplace necessitou de envolver um conjunto de testes que visavam verificar a consistência do mesmo, a forma como reage a tentativas de acesso não autorizadas e como se comporta em situações de acesso concorrente por parte de mais do que um utilizador autorizado, de forma simultânea.

Para tal, a ferramenta de testes escolhida foi o JMeter [14], que faz parte do projecto Jakarta da Apache Software Foundation.

O JMeter é uma ferramenta *open source* totalmente desenvolvida em Java.

É sobretudo utilizado na realização de testes de performance, de carga e de stress, embora a sua utilização também seja válida na execução de testes em serviços web e bases de dados. Disponibiliza uma interface gráfica que facilita a visualização e interpretação dos resultados dos testes, suporta *multithreading* que é controlado através de um *thread group* onde é possível configurar o número de *threads* a serem executadas, o número de execuções para cada uma e o tempo de intervalo entre cada execução. Existem ainda diversos componentes denominados *listeners* que recolhem e agregam os resultados obtidos através dos testes e a partir dos quais é possível a geração de gráficos e tabelas, com dados estatísticos.

Foi realizado um plano de testes, que visa ser transversal a todos os 3 serviços web disponibilizados através da construção do Mediador, de modo a que todos eles sejam testados em condições idênticas para que se possa estabelecer um termo de comparação entre eles e o seu desempenho.

Para cada serviço pretende-se testar a diferença do seu comportamento e tempos de resposta em situações em que a carga envolvida nas requisições seja mais baixa ou mais alta, assim como fazer variar o número de utilizadores que acedem ao serviço, de forma simultânea.

Por fim, pretende-se realizar uma avaliação global do sistema como um todo.

5.1 Plano de Testes

Serviços web

- Search (Objects);
- Fetch;
- Upload.

Variação da carga envolvida nas requisições
Serviço web – Search (Objects)
Requisição mais leve – Pesquisa de um objecto específico no repositório
Requisição mais pesada – Pesquisa de todos os objectos do repositório

Serviço web – Fetch
Requisição mais leve – *Download* de um conteúdo de pequena dimensão (~4Kb)
Requisição mais pesada – *Download* de um conteúdo de maior dimensão (~1.1Mb)

Serviço web – Upload
Requisição mais leve – Envio de um conteúdo de pequena dimensão (~13Kb)
Requisição mais pesada – Envio de um conteúdo de maior dimensão (~4,55Mb)

Na *Figura 13* encontra-se representada uma possível estrutura para um plano de testes a ser executado na ferramenta JMeter – semelhante à estrutura utilizada para os testes aos serviços de Pesquisa e Download.

A organização da estrutura do plano de testes não é propriamente rígida e pode variar de acordo com o utilizador que está a realizar os testes e com os objectivos dos testes em si, ou seja, aquilo que se pretende testar e o tipo de resultados que se pretendem observar.

No caso concreto dos testes realizados ao Mediador, uma vez que se pretendia sobretudo analisar os tempos de resposta forma os seus serviços

reagiam a requisições efectuadas por vários utilizadores em simultâneo, foi esta a estrutura do plano de testes escolhida.

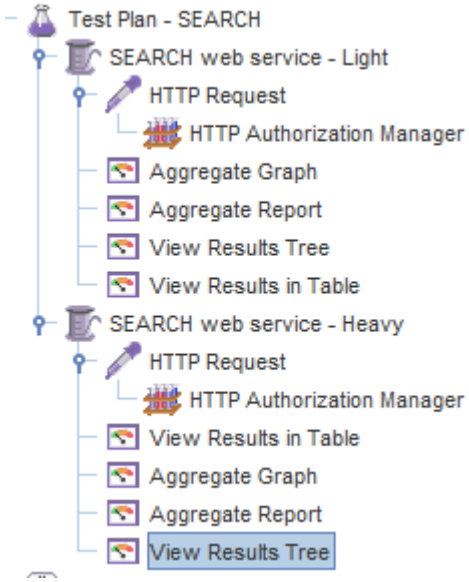


Figura 13: Estrutura de um possível plano de testes na ferramenta JMeter

Na *Figura 14* está representado o primeiro elemento da estrutura do plano de testes acima apresentada.

O elemento representado nesta figura é um *Thread Group*. Qualquer plano de testes na ferramenta JMeter tem obrigatoriamente de ter pelo menos um elemento destes. O *Thread Group* é o elemento de partida do plano de testes e é o elemento onde estão contidos todos os outros elementos do plano. É nele que são controladas as threads que serão criadas pela ferramenta para simular o acesso simultâneo de utilizadores a um determinado serviço. Cada thread representa um único utilizador. O *Ramp-Up Period* define o tempo em segundos, que o JMeter levará a criar as threads pedidas pelo utilizador, isto é, para um caso com 10 threads e um período de 20 segundos, a ferramenta JMeter irá criar uma thread de 2 em 2 segundos. Nos testes realizados para o Mediador, uma vez que se pretende que todas as threads sejam iniciadas ao mesmo tempo, o valor colocado neste campo foi 0 (zero) para todos os testes. O *Loop-Count* define o número de repetições do teste pretendidas.

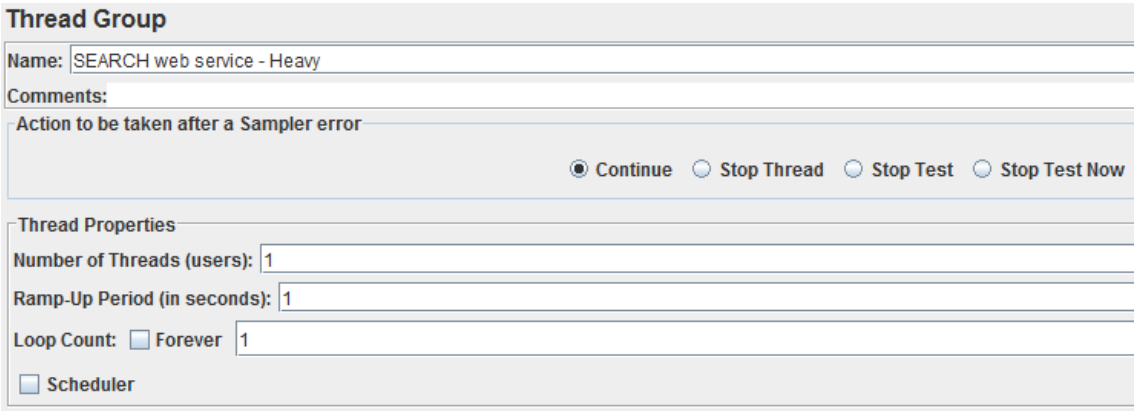


Figura 14: Exemplo de configuração de um Thread Group na ferramenta JMeter

Na *Figura 15* está representado o elemento seguinte do plano de testes. O elemento *HTTP Request*, à semelhança dos restantes elementos, encontra-se contido no elemento principal, o *Thread Group.*

É neste elemento que é configurada a requisição HTTP que se pretende realizar a um determinado serviço. É ainda possível configurar um conjunto de propriedades no painel deste elemento, como por exemplo o *Server Name or IP* e o porto da máquina onde está a correr a aplicação que se pretende testar, o protocolo utilizado na requisição, o método HTTP, a codificação do conteúdo, o caminho para o recurso sobre o qual vão incidir os testes e os parâmetros enviados com a requisição, entre outros.

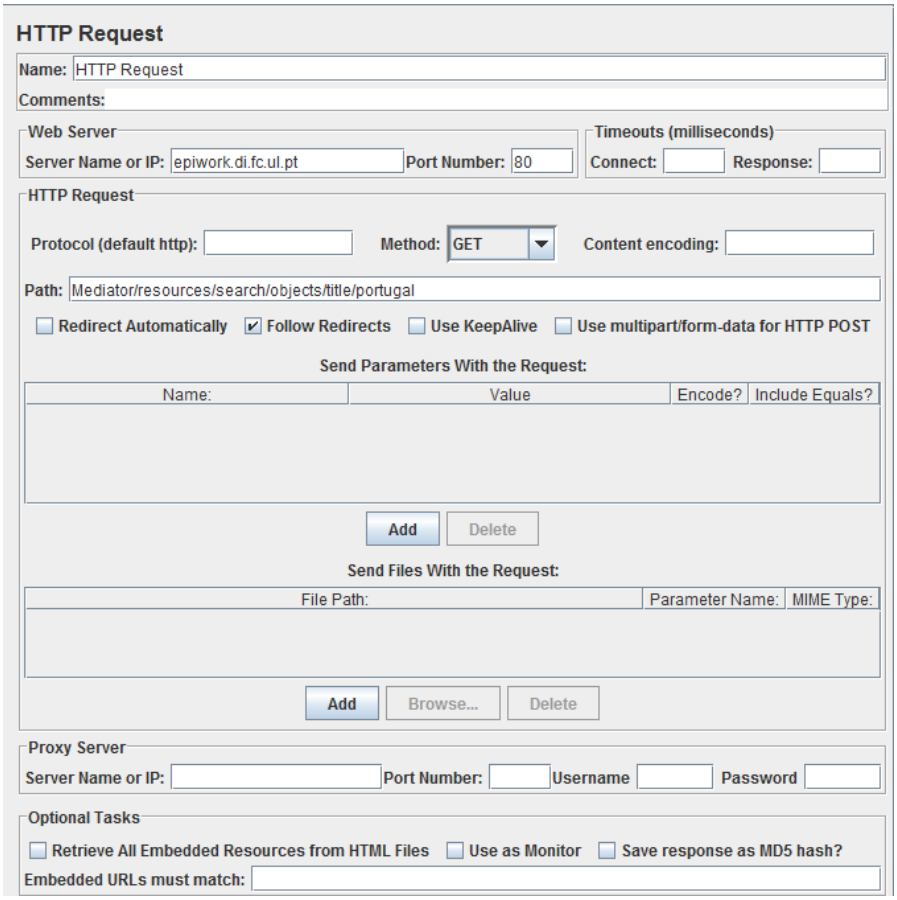


Figura 15: Exemplo de configuração de um elemento HTTP Request na ferramenta JMeter

O elemento seguinte é o *HTTP Authorization Manager (Figura 16)* que permite que o utilizador especifique uma ou mais credenciais de acesso, para as páginas web com acesso restrito e que utilizam como esquema de autenticação o *Basic Access Authentication HTTP*. Quando a ferramenta JMeter encontra uma aplicação que concretiza este esquema de autenticação, transmite os dados que são colocados neste elemento, obtendo o acesso à mesma.

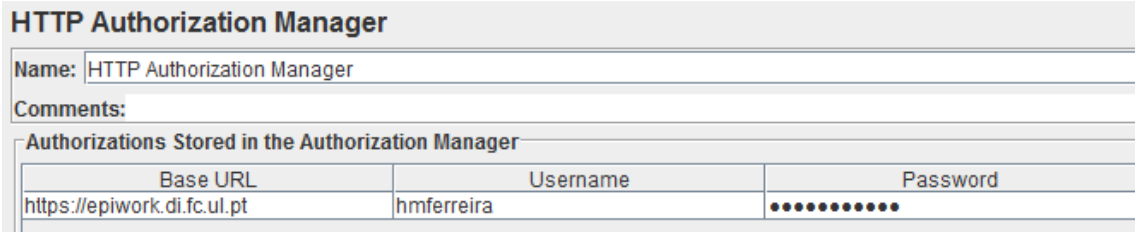


Figura 16: Exemplo de configuração de um elemento HTTP Authorization Manager na ferramenta JMeter

5.2 Testes e Resultados

Serviço web: Search

Tipo de pedido: Leve

Url:

https://epiwork.di.fc.ul.pt/Mediator/resources/search/objects/title/portugal

Número de *threads* (utilizadores): 1

Número de repetições: 5

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	19:11:39.799	SEARCH web service - Light 1-1	HTTP Request	706		2394
2	19:11:40.506	SEARCH web service - Light 1-1	HTTP Request	242		2394
3	19:11:40.749	SEARCH web service - Light 1-1	HTTP Request	271		2394
4	19:11:41.020	SEARCH web service - Light 1-1	HTTP Request	267		2394
5	19:11:41.288	SEARCH web service - Light 1-1	HTTP Request	254		2394

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Request	5	348	267	271	242	706	0,00%	2,9/sec	6,7
TOTAL	5	348	267	271	242	706	0,00%	2,9/sec	6,7

Figura 17: Teste 1, serviço web de pesquisa

Número de *threads* (utilizadores): 5

Número de repetições: 3

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	19:14:58.828	SEARCH web ser...	HTTP Request	1239		2394
2	19:14:58.826	SEARCH web ser...	HTTP Request	1264		2394
3	19:14:58.827	SEARCH web ser...	HTTP Request	1268		2394
4	19:14:58.828	SEARCH web ser...	HTTP Request	1272		2394
5	19:14:58.827	SEARCH web ser...	HTTP Request	1360		2394
6	19:15:00.068	SEARCH web ser...	HTTP Request	477		2394
7	19:15:00.092	SEARCH web ser...	HTTP Request	463		2394
8	19:15:00.190	SEARCH web ser...	HTTP Request	380		2394
9	19:15:00.101	SEARCH web ser...	HTTP Request	520		2394
10	19:15:00.096	SEARCH web ser...	HTTP Request	536		2394
11	19:15:00.571	SEARCH web ser...	HTTP Request	452		2394
12	19:15:00.546	SEARCH web ser...	HTTP Request	489		2394
13	19:15:00.556	SEARCH web ser...	HTTP Request	489		2394
14	19:15:00.622	SEARCH web ser...	HTTP Request	457		2394
15	19:15:00.634	SEARCH web ser...	HTTP Request	500		2394

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	15	680	623	945	434	1099	0,00%	7,2/sec	16,9
TOTAL	15	680	623	945	434	1099	0,00%	7,2/sec	16,9

Figura 18: Teste 2, serviço web de pesquisa

Número de *threads* (utilizadores): 15

Número de repetições: 2

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	03:06:11.511	SEARCH web service - Light 1-13	HTTP Request	1865		2394
2	03:06:11.513	SEARCH web service - Light 1-10	HTTP Request	1938		2394
3	03:06:11.512	SEARCH web service - Light 1-8	HTTP Request	2216		2394
4	03:06:11.510	SEARCH web service - Light 1-9	HTTP Request	2233		2394
5	03:06:11.512	SEARCH web service - Light 1-6	HTTP Request	2465		2394
6	03:06:11.508	SEARCH web service - Light 1-1	HTTP Request	2474		2394
7	03:06:11.509	SEARCH web service - Light 1-3	HTTP Request	2476		2394
8	03:06:11.512	SEARCH web service - Light 1-4	HTTP Request	2474		2394
9	03:06:11.514	SEARCH web service - Light 1-15	HTTP Request	2475		2394
10	03:06:11.510	SEARCH web service - Light 1-7	HTTP Request	2481		2394
11	03:06:11.512	SEARCH web service - Light 1-2	HTTP Request	2482		2394
12	03:06:11.511	SEARCH web service - Light 1-11	HTTP Request	2576		2394
13	03:06:11.510	SEARCH web service - Light 1-5	HTTP Request	2612		2394
14	03:06:11.513	SEARCH web service - Light 1-14	HTTP Request	2651		2394
15	03:06:13.453	SEARCH web service - Light 1-10	HTTP Request	713		2394
16	03:06:11.511	SEARCH web service - Light 1-12	HTTP Request	2658		2394
17	03:06:13.377	SEARCH web service - Light 1-13	HTTP Request	833		2394
18	03:06:13.729	SEARCH web service - Light 1-8	HTTP Request	830		2394
19	03:06:13.987	SEARCH web service - Light 1-4	HTTP Request	782		2394
20	03:06:13.744	SEARCH web service - Light 1-9	HTTP Request	1090		2394
21	03:06:13.992	SEARCH web service - Light 1-7	HTTP Request	1049		2394
22	03:06:13.982	SEARCH web service - Light 1-1	HTTP Request	1072		2394
23	03:06:14.088	SEARCH web service - Light 1-11	HTTP Request	1163		2394
24	03:06:14.169	SEARCH web service - Light 1-12	HTTP Request	1084		2394
25	03:06:13.994	SEARCH web service - Light 1-2	HTTP Request	1262		2394
26	03:06:13.990	SEARCH web service - Light 1-15	HTTP Request	1465		2394
27	03:06:14.165	SEARCH web service - Light 1-14	HTTP Request	1292		2394
28	03:06:14.122	SEARCH web service - Light 1-5	HTTP Request	1337		2394
29	03:06:13.978	SEARCH web service - Light 1-6	HTTP Request	1484		2394
30	03:06:13.986	SEARCH web service - Light 1-3	HTTP Request	3203		2394

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Request	30	1824	1865	2612	713	3203	0,00%	5,3/sec	12,3
TOTAL	30	1824	1865	2612	713	3203	0,00%	5,3/sec	12,3

Figura 19: Teste 3, serviço web de pesquisa

Tipo de pedido: Pesado

Url: https://epiwork.di.fc.ul.pt/Mediator/resources/search/objects/

Número de *threads* (utilizadores): 1

Número de repetições: 5

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	18:36:18.325	SEARCH web service - Heavy 1-1	HTTP Request	21519		204202
2	18:36:39.845	SEARCH web service - Heavy 1-1	HTTP Request	21154		204202
3	18:37:01.000	SEARCH web service - Heavy 1-1	HTTP Request	34847		204202
4	18:37:35.848	SEARCH web service - Heavy 1-1	HTTP Request	20598		204202
5	18:37:56.447	SEARCH web service - Heavy 1-1	HTTP Request	20532		204202

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Request	5	23730	21154	21519	20532	34847	0,00%	2,5/min	8,4
TOTAL	5	23730	21154	21519	20532	34847	0,00%	2,5/min	8,4

Figura 20: Teste 4, serviço web de pesquisa

Número de *threads* (utilizadores): 5

Número de repetições: 3

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	18:44:57.660	SEARCH web ser...	HTTP Request	50204		204202
2	18:44:57.661	SEARCH web ser...	HTTP Request	51085		204202
3	18:44:57.660	SEARCH web ser...	HTTP Request	51148		204202
4	18:44:57.660	SEARCH web ser...	HTTP Request	51175		204202
5	18:44:57.661	SEARCH web ser...	HTTP Request	51181		204202
6	18:45:48.836	SEARCH web ser...	HTTP Request	39360		204202
7	18:45:47.865	SEARCH web ser...	HTTP Request	41186		204202
8	18:45:48.843	SEARCH web ser...	HTTP Request	40242		204202
9	18:45:48.810	SEARCH web ser...	HTTP Request	40392		204202
10	18:45:48.748	SEARCH web ser...	HTTP Request	40482		204202
11	18:46:29.052	SEARCH web ser...	HTTP Request	41083		204202
12	18:46:29.086	SEARCH web ser...	HTTP Request	41054		204202
13	18:46:29.203	SEARCH web ser...	HTTP Request	40961		204202
14	18:46:28.196	SEARCH web ser...	HTTP Request	42060		204202
15	18:46:29.230	SEARCH web ser...	HTTP Request	41060		204202

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	15	44178	41083	51148	39360	51181	0,00%	6,8/min	21,1
TOTAL	15	44178	41083	51148	39360	51181	0,00%	6,8/min	21,1

Figura 21: Teste 5, serviço web de pesquisa

Número de *threads* (utilizadores): 15

Número de repetições: 2

Sample #	Start Time	Thread Name	Label	Sample Time(...	Status	Bytes
1	20:03:08.580	SEARCH web service - Heavy 1-7	HTTP Request	111894		204202
2	20:03:08.583	SEARCH web service - Heavy 1-14	HTTP Request	112095		204202
3	20:03:08.551	SEARCH web service - Heavy 1-1	HTTP Request	112587		204202
4	20:03:08.599	SEARCH web service - Heavy 1-3	HTTP Request	112603		204202
5	20:03:08.598	SEARCH web service - Heavy 1-11	HTTP Request	112815		204202
6	20:03:08.580	SEARCH web service - Heavy 1-9	HTTP Request	112877		204202
7	20:03:08.580	SEARCH web service - Heavy 1-8	HTTP Request	113469		204202
8	20:03:08.598	SEARCH web service - Heavy 1-10	HTTP Request	113504		204202
9	20:03:08.580	SEARCH web service - Heavy 1-6	HTTP Request	113710		204202
10	20:03:08.598	SEARCH web service - Heavy 1-15	HTTP Request	114130		204202
11	20:03:08.579	SEARCH web service - Heavy 1-5	HTTP Request	114153		204202
12	20:03:08.583	SEARCH web service - Heavy 1-12	HTTP Request	115316		204202
13	20:03:08.598	SEARCH web service - Heavy 1-13	HTTP Request	115574		204202
14	20:03:08.557	SEARCH web service - Heavy 1-4	HTTP Request	115842		204202
15	20:03:08.553	SEARCH web service - Heavy 1-2	HTTP Request	116844		204202
16	20:05:04.399	SEARCH web service - Heavy 1-4	HTTP Request	107351		204202
17	20:05:00.475	SEARCH web service - Heavy 1-7	HTTP Request	111496		204202
18	20:05:02.294	SEARCH web service - Heavy 1-6	HTTP Request	110107		204202
19	20:05:01.458	SEARCH web service - Heavy 1-9	HTTP Request	110958		204202
20	20:05:02.050	SEARCH web service - Heavy 1-8	HTTP Request	110733		204202
21	20:05:01.413	SEARCH web service - Heavy 1-11	HTTP Request	111487		204202
22	20:05:05.398	SEARCH web service - Heavy 1-2	HTTP Request	107633		204202
23	20:05:02.103	SEARCH web service - Heavy 1-10	HTTP Request	110965		204202
24	20:05:02.733	SEARCH web service - Heavy 1-5	HTTP Request	110741		204202
25	20:05:01.140	SEARCH web service - Heavy 1-1	HTTP Request	112359		204202
26	20:05:01.203	SEARCH web service - Heavy 1-3	HTTP Request	113863		204202
27	20:05:04.173	SEARCH web service - Heavy 1-13	HTTP Request	110981		204202
28	20:05:02.728	SEARCH web service - Heavy 1-15	HTTP Request	112485		204202
29	20:05:03.900	SEARCH web service - Heavy 1-12	HTTP Request	111316		204202
30	20:05:00.684	SEARCH web service - Heavy 1-14	HTTP Request	114745		204202

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	30	112487	112485	115316	107351	116844	0,00%	7,9/min	23,7
TOTAL	30	112487	112485	115316	107351	116844	0,00%	7,9/min	23,7

Figura 22: Teste 6, serviço web de pesquisa

Número de *threads* (utilizadores): 20

Número de repetições: 1

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	03:09:27.147	SEARCH web service - Heavy 1-7	HTTP Request	9041		2379
2	03:09:27.149	SEARCH web service - Heavy 1-12	HTTP Request	13733		2379
3	03:09:27.147	SEARCH web service - Heavy 1-9	HTTP Request	14265		2379
4	03:09:27.147	SEARCH web service - Heavy 1-3	HTTP Request	120118		861
5	03:09:27.148	SEARCH web service - Heavy 1-13	HTTP Request	120293		861
6	03:09:27.147	SEARCH web service - Heavy 1-1	HTTP Request	120311		861
7	03:09:27.149	SEARCH web service - Heavy 1-10	HTTP Request	120310		861
8	03:09:27.148	SEARCH web service - Heavy 1-4	HTTP Request	120323		861
9	03:09:27.148	SEARCH web service - Heavy 1-17	HTTP Request	120331		861
10	03:09:27.148	SEARCH web service - Heavy 1-6	HTTP Request	120331		861
11	03:09:27.138	SEARCH web service - Heavy 1-2	HTTP Request	120345		861
12	03:09:27.149	SEARCH web service - Heavy 1-18	HTTP Request	120334		861
13	03:09:27.147	SEARCH web service - Heavy 1-11	HTTP Request	120338		861
14	03:09:27.150	SEARCH web service - Heavy 1-20	HTTP Request	120391		861
15	03:09:27.149	SEARCH web service - Heavy 1-16	HTTP Request	120459		861
16	03:09:27.147	SEARCH web service - Heavy 1-5	HTTP Request	123073		861
17	03:09:27.140	SEARCH web service - Heavy 1-19	HTTP Request	123079		861
18	03:09:27.149	SEARCH web service - Heavy 1-14	HTTP Request	123083		861
19	03:09:27.148	SEARCH web service - Heavy 1-15	HTTP Request	123352		861
20	03:09:27.148	SEARCH web service - Heavy 1-8	HTTP Request	126043		861

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Request	20	104977	120331	123083	9041	126043	85,00%	9,5/min	,2
TOTAL	20	104977	120331	123083	9041	126043	85,00%	9,5/min	,2

Figura 23: Teste adicional, serviço web de pesquisa

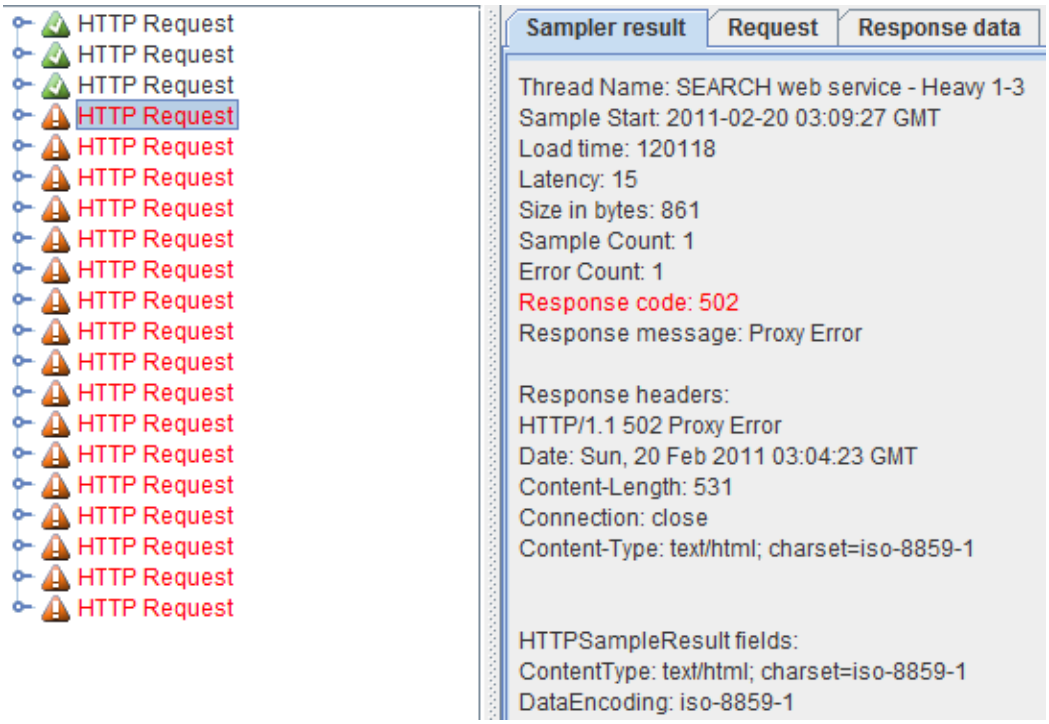


Figura 24: Detalhe do erro no teste ao serviço web de pesquisa

Análise aos resultados para o serviço web de Pesquisa:

Nos testes realizados para este serviço web disponibilizado pelo Mediador, representados pelas *Figuras Figura* 17*Figura 22*, foi possível verificar a inexistência de falhas na disponibilidade do serviço para atender as requisições, mesmo quando estas eram executadas por várias threads em simultâneo.

Esta disponibilidade do serviço foi visível tanto para os casos em que as requisições envolviam uma carga mais leve (representada pela pesquisa de um conteúdo específico do repositório), como para o caso dos pedidos com carga mais elevada (representados pela pesquisa de todos os conteúdos existentes no repositório).

Com o aumento do número de *threads* que acediam ao serviço de forma concorrente, aumentava também o tempo que o serviço demorava a responder, embora se possa considerar que este aumento do tempo médio de resposta seja normal, uma vez que este acontece de forma praticamente proporcional ao do tamanho da resposta devolvida. Esta afirmação é confirmada pelo gráfico

apresentado na *Figura 25* onde é possível ter uma melhor percepção da forma como os tempos médios de resposta às requisições, variam em função da carga da requisição (Leve/Pesada) e do número de utilizadores (1, 5, 15) que a concretizam.

No entanto, como qualquer outra aplicação existente na web, também o Mediador é falível. Prova disto mesmo é o teste adicional que foi realizado ao serviço web de pesquisa e que se encontra representado na *Figura 23*. Neste teste simulou-se a requisição do serviço concretizada por 20 *threads* em simultâneo. Na figura é perceptível que praticamente todas as threads retornaram um erro por parte do serviço. Na *Figura 24* é visível o detalhe do erro (HTTP 502 Proxy Error), provocado pelo envio de uma resposta inválida por parte do servidor onde se encontra alojado o Mediador e que trata o pedido. Esta resposta inválida deve-se à sobrecarga a que o serviço foi sujeito e como tal, o mesmo não conseguiu responder correctamente a todas as requisições que lhe foram feitas em simultâneo.

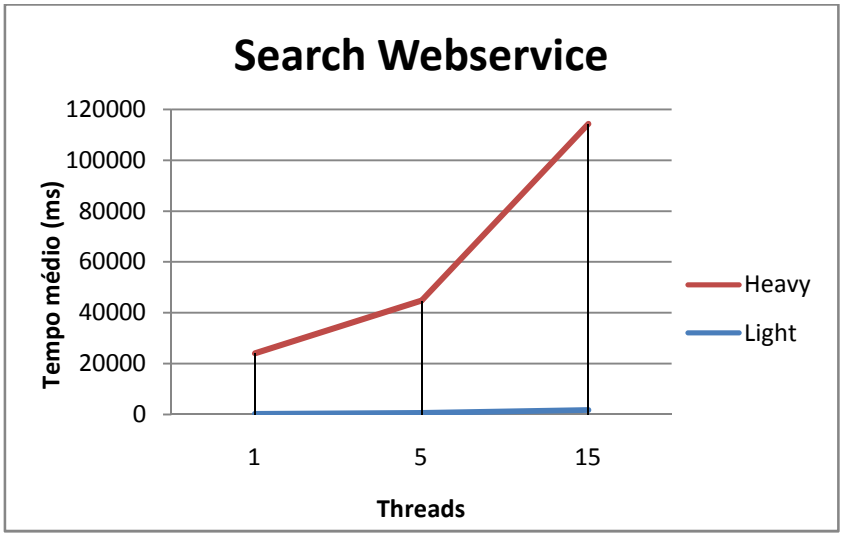


Figura 25: Gráfico de desempenho do serviço web de Pesquisa

Serviço web: Fetch

Tipo de pedido: Leve

Url: https://epiwork.di.fc.ul.pt/Mediator/resources/fetch/168

Tamanho do ficheiro: ~ 4Kb

Número de *threads* (utilizadores): 1

Número de repetições: 5

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	20:53:29.798	FETCH web servi...	HTTP Request	506		3752
2	20:53:30.305	FETCH web servi...	HTTP Request	177		3752
3	20:53:30.483	FETCH web servi...	HTTP Request	486		3752
4	20:53:30.973	FETCH web servi...	HTTP Request	174		3752
5	20:53:31.149	FETCH web servi...	HTTP Request	232		3752

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	5	315	232	486	174	506	0,00%	3,2/sec	11,6
TOTAL	5	315	232	486	174	506	0,00%	3,2/sec	11,6

Figura 26: Teste 1, serviço web de download

Número de *threads* (utilizadores): 5

Número de repetições: 3

Sample #	Start Time	Thread Name	Label	Sample Time(...	Status	Bytes
1	00:06:48.410	FETCH web service - Light 1-5	HTTP Request	337		3752
2	00:06:48.408	FETCH web service - Light 1-1	HTTP Request	386		3752
3	00:06:48.410	FETCH web service - Light 1-3	HTTP Request	388		3752
4	00:06:48.409	FETCH web service - Light 1-4	HTTP Request	393		3752
5	00:06:48.409	FETCH web service - Light 1-2	HTTP Request	602		3752
6	00:06:48.748	FETCH web service - Light 1-5	HTTP Request	284		3752
7	00:06:48.798	FETCH web service - Light 1-3	HTTP Request	325		3752
8	00:06:48.803	FETCH web service - Light 1-4	HTTP Request	324		3752
9	00:06:48.795	FETCH web service - Light 1-1	HTTP Request	340		3752
10	00:06:49.012	FETCH web service - Light 1-2	HTTP Request	255		3752
11	00:06:49.032	FETCH web service - Light 1-5	HTTP Request	249		3752
12	00:06:49.128	FETCH web service - Light 1-4	HTTP Request	471		3752
13	00:06:49.136	FETCH web service - Light 1-1	HTTP Request	492		3752
14	00:06:49.268	FETCH web service - Light 1-2	HTTP Request	475		3752
15	00:06:49.124	FETCH web service - Light 1-3	HTTP Request	622		3752

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	15	396	386	492	249	622	0,00%	11,2/sec	41,1
TOTAL	15	396	386	492	249	622	0,00%	11,2/sec	41,1

Figura 27: Teste 2, serviço web de download

Número de *threads* (utilizadores): 15

Número de repetições: 2

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	00:13:35.460	FETCH web service - Light 1-4	HTTP Request	490		3752
2	00:13:35.466	FETCH web service - Light 1-13	HTTP Request	995		3752
3	00:13:35.483	FETCH web service - Light 1-14	HTTP Request	1052		3752
4	00:13:35.476	FETCH web service - Light 1-10	HTTP Request	1085		3752
5	00:13:35.471	FETCH web service - Light 1-8	HTTP Request	1141		3752
6	00:13:35.463	FETCH web service - Light 1-11	HTTP Request	1171		3752
7	00:13:35.461	FETCH web service - Light 1-9	HTTP Request	1210		3752
8	00:13:35.467	FETCH web service - Light 1-15	HTTP Request	1270		3752
9	00:13:35.460	FETCH web service - Light 1-6	HTTP Request	1288		3752
10	00:13:35.476	FETCH web service - Light 1-12	HTTP Request	1290		3752
11	00:13:35.453	FETCH web service - Light 1-5	HTTP Request	1326		3752
12	00:13:35.449	FETCH web service - Light 1-1	HTTP Request	1335		3752
13	00:13:35.451	FETCH web service - Light 1-3	HTTP Request	1337		3752
14	00:13:35.951	FETCH web service - Light 1-4	HTTP Request	840		3752
15	00:13:35.460	FETCH web service - Light 1-7	HTTP Request	1625		3752
16	00:13:35.451	FETCH web service - Light 1-2	HTTP Request	1733		3752
17	00:13:36.462	FETCH web service - Light 1-13	HTTP Request	973		3752
18	00:13:36.535	FETCH web service - Light 1-14	HTTP Request	904		3752
19	00:13:36.673	FETCH web service - Light 1-9	HTTP Request	896		3752
20	00:13:36.749	FETCH web service - Light 1-6	HTTP Request	860		3752
21	00:13:36.562	FETCH web service - Light 1-10	HTTP Request	1065		3752
22	00:13:36.780	FETCH web service - Light 1-5	HTTP Request	877		3752
23	00:13:36.635	FETCH web service - Light 1-11	HTTP Request	1038		3752
24	00:13:36.788	FETCH web service - Light 1-3	HTTP Request	1146		3752
25	00:13:36.613	FETCH web service - Light 1-8	HTTP Request	1606		3225
26	00:13:36.738	FETCH web service - Light 1-15	HTTP Request	1606		3225
27	00:13:36.784	FETCH web service - Light 1-1	HTTP Request	1685		3752
28	00:13:37.086	FETCH web service - Light 1-7	HTTP Request	1387		3752
29	00:13:37.184	FETCH web service - Light 1-2	HTTP Request	1293		3752
30	00:13:36.766	FETCH web service - Light 1-12	HTTP Request	1715		3752

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	30	1207	1171	1625	490	1733	0,00%	9,9/sec	35,9
TOTAL	30	1207	1171	1625	490	1733	0,00%	9,9/sec	35,9

Figura 28: Teste 3, serviço web de download

Tipo de pedido: Pesado

Url: https://epiwork.di.fc.ul.pt/Mediator/resources/fetch/218

Tamanho do ficheiro: ~1,1Mb

Número de *threads* (utilizadores): 1

Número de repetições: 5

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	21:33:07.918	FETCH web servi...	HTTP Request	2255		1149785
2	21:33:10.174	FETCH web servi...	HTTP Request	1896		1149785
3	21:33:12.071	FETCH web servi...	HTTP Request	1906		1149785
4	21:33:13.977	FETCH web servi...	HTTP Request	2050		1149785
5	21:33:16.028	FETCH web servi...	HTTP Request	1990		1149785

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	5	2019	1990	2050	1896	2255	0,00%	29,7/min	555,9
TOTAL	5	2019	1990	2050	1896	2255	0,00%	29,7/min	555,9

Figura 29: Teste 4, serviço web de download

Número de *threads* (utilizadores): 5

Número de repetições: 3

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	02:39:08.130	FETCH web service - Heavy 1-4	HTTP Request	5612		1149785
2	02:39:08.129	FETCH web service - Heavy 1-1	HTTP Request	5886		1149785
3	02:39:08.130	FETCH web service - Heavy 1-3	HTTP Request	5893		1149785
4	02:39:08.130	FETCH web service - Heavy 1-2	HTTP Request	5963		1149785
5	02:39:08.131	FETCH web service - Heavy 1-5	HTTP Request	6248		1149785
6	02:39:14.016	FETCH web service - Heavy 1-1	HTTP Request	5055		1149785
7	02:39:13.743	FETCH web service - Heavy 1-4	HTTP Request	5615		1149785
8	02:39:14.094	FETCH web service - Heavy 1-2	HTTP Request	5351		1149785
9	02:39:14.023	FETCH web service - Heavy 1-3	HTTP Request	5832		1149785
10	02:39:14.380	FETCH web service - Heavy 1-5	HTTP Request	5624		1149785
11	02:39:19.446	FETCH web service - Heavy 1-2	HTTP Request	5497		1149785
12	02:39:19.359	FETCH web service - Heavy 1-4	HTTP Request	6007		1149785
13	02:39:19.072	FETCH web service - Heavy 1-1	HTTP Request	6412		1149785
14	02:39:20.005	FETCH web service - Heavy 1-5	HTTP Request	5528		1149785
15	02:39:19.856	FETCH web service - Heavy 1-3	HTTP Request	5747		1149785

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	15	5751	5747	6007	5055	6412	0,00%	51,5/min	963,9
TOTAL	15	5751	5747	6007	5055	6412	0,00%	51,5/min	963,9

Figura 30: Teste 5, serviço web de download

Número de *threads* (utilizadores): 10

Número de repetições: 2

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	00:27:38.520	FETCH web service - Heavy 1-6	HTTP Request	2500		1149785
2	00:27:38.511	FETCH web service - Heavy 1-13	HTTP Request	10305		1149785
3	00:27:38.508	FETCH web service - Heavy 1-5	HTTP Request	11434		1149785
4	00:27:38.513	FETCH web service - Heavy 1-14	HTTP Request	18103		1149785
5	00:27:38.507	FETCH web service - Heavy 1-3	HTTP Request	22376		1149785
6	00:27:41.021	FETCH web service - Heavy 1-6	HTTP Request	20037		1149785
7	00:27:38.511	FETCH web service - Heavy 1-4	HTTP Request	23042		1149785
8	00:27:38.505	FETCH web service - Heavy 1-1	HTTP Request	24331		1149785
9	00:27:38.511	FETCH web service - Heavy 1-2	HTTP Request	24646		1149785
10	00:27:38.512	FETCH web service - Heavy 1-12	HTTP Request	25344		1149785
11	00:27:38.510	FETCH web service - Heavy 1-11	HTTP Request	25973		1149785
12	00:27:38.512	FETCH web service - Heavy 1-10	HTTP Request	26020		1149785
13	00:27:38.510	FETCH web service - Heavy 1-9	HTTP Request	26463		1149785
14	00:27:49.943	FETCH web service - Heavy 1-5	HTTP Request	15519		1149785
15	00:27:38.511	FETCH web service - Heavy 1-15	HTTP Request	27100		1149785
16	00:28:04.484	FETCH web service - Heavy 1-11	HTTP Request	1290		1149785
17	00:27:38.509	FETCH web service - Heavy 1-7	HTTP Request	27354		1149785
18	00:27:38.512	FETCH web service - Heavy 1-8	HTTP Request	28028		1149785
19	00:27:48.817	FETCH web service - Heavy 1-13	HTTP Request	18873		1149785
20	00:27:56.617	FETCH web service - Heavy 1-14	HTTP Request	11174		1149785
21	00:28:00.884	FETCH web service - Heavy 1-3	HTTP Request	8433		1149785
22	00:28:02.837	FETCH web service - Heavy 1-1	HTTP Request	9456		1149785
23	00:28:01.553	FETCH web service - Heavy 1-4	HTTP Request	11375		1149785
24	00:28:03.158	FETCH web service - Heavy 1-2	HTTP Request	11004		1149785
25	00:28:03.857	FETCH web service - Heavy 1-12	HTTP Request	10738		1149785
26	00:28:04.974	FETCH web service - Heavy 1-9	HTTP Request	10356		1149785
27	00:28:05.611	FETCH web service - Heavy 1-15	HTTP Request	9890		1149785
28	00:28:04.533	FETCH web service - Heavy 1-10	HTTP Request	11191		1149785
29	00:28:05.864	FETCH web service - Heavy 1-7	HTTP Request	9882		1149785
30	00:28:06.540	FETCH web service - Heavy 1-8	HTTP Request	9486		1149785

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	30	16390	11434	26463	1290	28028	0,00%	48,0/min	851,8
TOTAL	30	16390	11434	26463	1290	28028	0,00%	48,0/min	851,8

Figura 31: Teste 6, serviço web de download

Análise aos resultados para o serviço web de Download:

À semelhança do que já havia acontecido nos testes para o serviço web de Pesquisa, também os testes para o serviço web de Download decorreu sem problemas e sem falhas no que diz respeito à disponibilidade do serviço e à sua capacidade para responder às requisições que lhe eram feitas, mesmo que de forma simultânea, como é possível comprovar pela sequência resultados apresentados desde a *Figura 26* à *Figura 31*.

O tempo de resposta variou consoante a carga das requisições e com o número de requisições que eram efectuadas em simultâneo. A *Figura 31* é disto um bom exemplo, uma vez que é possível verificar que a diferença entre o tempo de resposta a requisições feitas por threads distintas chega a ser de mais de 20 segundos, o que reflecte os picos de processamento do servidor em determinados momentos, durante os quais devolve respostas mais demoradas.

No gráfico da *Figura 32* podemos observar uma evolução do tempo médio de respostas, em relação ao número de *threads* que concretizavam as requisições, bastante semelhante ao que já havia acontecido no gráfico para o serviço web de Pesquisa. O tempo de resposta é significativamente superior na requisição mais pesada e esta diferença é cada vez mais notória à medida que o número de utilizadores aumenta.

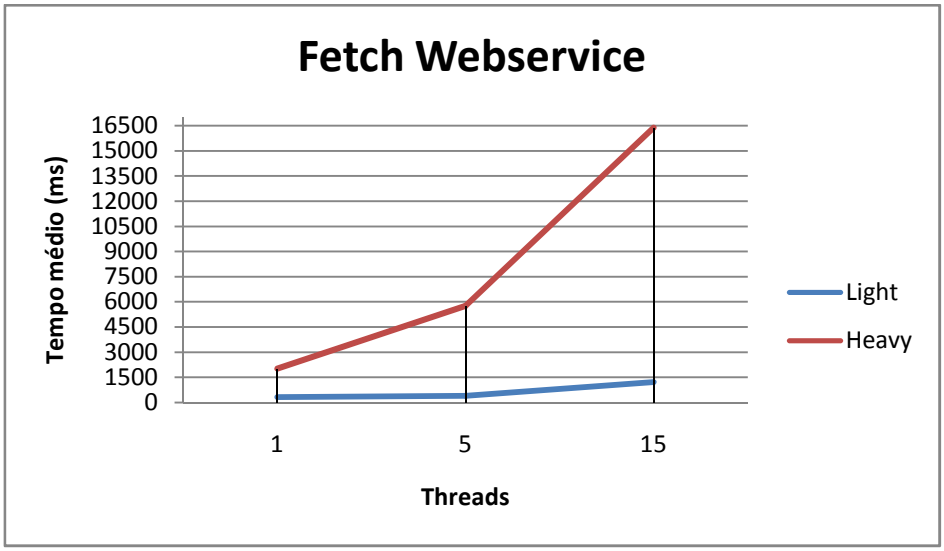


Figura 32: Gráfico de desempenho do serviço web de Download

Serviço web: Upload

Tipo de pedido: Leve

Url: https://epiwork.di.fc.ul.pt/Mediator/resources/upload

Tamanho do ficheiro: ~13Kb

Número de *threads* (utilizadores): 1

Número de repetições: 5

Sample #	Start Time	Thread Name	Label	Sample Time(m...	Status	Bytes
1	01:02:28.640	UPLOAD web service - Light 1-1	HTTP Request	20		323
2	01:02:28.660	UPLOAD web service - Light 1-1	HTTP Request	20		323
3	01:02:28.680	UPLOAD web service - Light 1-1	HTTP Request	19		323
4	01:02:28.700	UPLOAD web service - Light 1-1	HTTP Request	26		323
5	01:02:28.726	UPLOAD web service - Light 1-1	HTTP Request	21		323

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	5	21	20	21	19	26	0,00%	46,7/sec	14,7
TOTAL	5	21	20	21	19	26	0,00%	46,7/sec	14,7

Figura 33: Teste 1, serviço web de upload

Número de *threads* (utilizadores): 5

Número de repetições: 3

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	01:32:03.273	UPLOAD web ser...	HTTP Request	47		323
2	01:32:03.272	UPLOAD web ser...	HTTP Request	55		323
3	01:32:03.272	UPLOAD web ser...	HTTP Request	61		323
4	01:32:03.272	UPLOAD web ser...	HTTP Request	71		323
5	01:32:03.272	UPLOAD web ser...	HTTP Request	75		323
6	01:32:03.328	UPLOAD web ser...	HTTP Request	51		323
7	01:32:03.334	UPLOAD web ser...	HTTP Request	52		323
8	01:32:03.321	UPLOAD web ser...	HTTP Request	75		323
9	01:32:03.343	UPLOAD web ser...	HTTP Request	60		323
10	01:32:03.348	UPLOAD web ser...	HTTP Request	62		323
11	01:32:03.380	UPLOAD web ser...	HTTP Request	57		323
12	01:32:03.387	UPLOAD web ser...	HTTP Request	60		323
13	01:32:03.397	UPLOAD web ser...	HTTP Request	61		323
14	01:32:03.404	UPLOAD web ser...	HTTP Request	61		323
15	01:32:03.411	UPLOAD web ser...	HTTP Request	60		323

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	15	60	60	71	47	75	0,00%	75,4/sec	23,8
TOTAL	15	60	60	71	47	75	0,00%	75,4/sec	23,8

Figura 34: Teste 2, serviço web de upload

Número de *threads* (utilizadores): 15

Número de repetições: 2

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	01:34:41.992	UPLOAD web ser...	HTTP Request	89		323
2	01:34:42.022	UPLOAD web ser...	HTTP Request	96		323
3	01:34:42.020	UPLOAD web ser...	HTTP Request	102		323
4	01:34:42.022	UPLOAD web ser...	HTTP Request	107		323
5	01:34:42.001	UPLOAD web ser...	HTTP Request	135		323
6	01:34:42.000	UPLOAD web ser...	HTTP Request	142		323
7	01:34:42.021	UPLOAD web ser...	HTTP Request	128		323
8	01:34:41.999	UPLOAD web ser...	HTTP Request	157		323
9	01:34:41.997	UPLOAD web ser...	HTTP Request	166		323
10	01:34:42.021	UPLOAD web ser...	HTTP Request	149		323
11	01:34:42.021	UPLOAD web ser...	HTTP Request	159		323
12	01:34:42.018	UPLOAD web ser...	HTTP Request	169		323
13	01:34:41.996	UPLOAD web ser...	HTTP Request	198		323
14	01:34:41.994	UPLOAD web ser...	HTTP Request	206		323
15	01:34:42.021	UPLOAD web ser...	HTTP Request	190		323
16	01:34:42.082	UPLOAD web ser...	HTTP Request	183		323
17	01:34:42.119	UPLOAD web ser...	HTTP Request	177		323
18	01:34:42.123	UPLOAD web ser...	HTTP Request	183		323
19	01:34:42.137	UPLOAD web ser...	HTTP Request	176		323
20	01:34:42.130	UPLOAD web ser...	HTTP Request	190		323
21	01:34:42.143	UPLOAD web ser...	HTTP Request	184		323
22	01:34:42.150	UPLOAD web ser...	HTTP Request	183		323
23	01:34:42.157	UPLOAD web ser...	HTTP Request	183		323
24	01:34:42.171	UPLOAD web ser...	HTTP Request	177		323
25	01:34:42.164	UPLOAD web ser...	HTTP Request	190		323
26	01:34:42.181	UPLOAD web ser...	HTTP Request	180		323
27	01:34:42.195	UPLOAD web ser...	HTTP Request	172		323
28	01:34:42.188	UPLOAD web ser...	HTTP Request	190		323
29	01:34:42.201	UPLOAD web ser...	HTTP Request	184		323
30	01:34:42.212	UPLOAD web ser...	HTTP Request	182		323

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	30	164	177	190	89	206	0,00%	74,6/sec	23,5
TOTAL	30	164	177	190	89	206	0,00%	74,6/sec	23,5

Figura 35: Teste 3, serviço web de upload

Tipo de pedido: Leve

Url: https://epiwork.di.fc.ul.pt/Mediator/resources/upload

Tamanho do ficheiro: ~4,5Mb

Número de *threads* (utilizadores): 1

Número de repetições: 5

Sample #	Start Time	Thread Name	Label	Sample Time(...	Status	Bytes
1	02:07:52.637	UPLOAD web service - Light 1-1	HTTP Request	4087		323
2	02:07:56.725	UPLOAD web service - Light 1-1	HTTP Request	4084		323
3	02:08:00.810	UPLOAD web service - Light 1-1	HTTP Request	4075		323
4	02:08:04.886	UPLOAD web service - Light 1-1	HTTP Request	4076		323
5	02:08:08.963	UPLOAD web service - Light 1-1	HTTP Request	4059		323

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput
HTTP Requ...	5	4076	4076	4084	4059	4087	0,00%	14,7/min
TOTAL	5	4076	4076	4084	4059	4087	0,00%	14,7/min

Figura 36: Teste 4, serviço web de upload

Número de *threads* (utilizadores): 5

Número de repetições: 3

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	02:14:03.743	UPLOAD web service - Light 1-4	HTTP Request	16449		323
2	02:14:03.726	UPLOAD web service - Light 1-2	HTTP Request	18751		323
3	02:14:03.741	UPLOAD web service - Light 1-1	HTTP Request	20214		323
4	02:14:03.733	UPLOAD web service - Light 1-5	HTTP Request	20497		323
5	02:14:03.734	UPLOAD web service - Light 1-3	HTTP Request	21588		323
6	02:14:20.193	UPLOAD web service - Light 1-4	HTTP Request	20346		323
7	02:14:24.231	UPLOAD web service - Light 1-5	HTTP Request	18646		323
8	02:14:22.478	UPLOAD web service - Light 1-2	HTTP Request	21228		323
9	02:14:25.322	UPLOAD web service - Light 1-3	HTTP Request	18675		323
10	02:14:23.956	UPLOAD web service - Light 1-1	HTTP Request	23302		323
11	02:14:42.877	UPLOAD web service - Light 1-5	HTTP Request	18961		323
12	02:14:40.539	UPLOAD web service - Light 1-4	HTTP Request	21484		323
13	02:14:43.707	UPLOAD web service - Light 1-2	HTTP Request	18777		323
14	02:14:47.259	UPLOAD web service - Light 1-1	HTTP Request	16904		323
15	02:14:43.998	UPLOAD web service - Light 1-3	HTTP Request	20262		323

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	15	19738	20214	21484	16449	23302	0,00%	14,9/min	,1
TOTAL	15	19738	20214	21484	16449	23302	0,00%	14,9/min	,1

Figura 37: Teste 5, serviço web de upload

Número de *threads* (utilizadores): 10

Número de repetições: 2

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	02:49:21.251	UPLOAD web ser...	HTTP Request	31881		323
2	02:49:21.253	UPLOAD web ser...	HTTP Request	35071		323
3	02:49:21.252	UPLOAD web ser...	HTTP Request	35243		323
4	02:49:21.236	UPLOAD web ser...	HTTP Request	36964		323
5	02:49:21.254	UPLOAD web ser...	HTTP Request	38842		323
6	02:49:21.234	UPLOAD web ser...	HTTP Request	39437		323
7	02:49:21.232	UPLOAD web ser...	HTTP Request	40751		323
8	02:49:21.231	UPLOAD web ser...	HTTP Request	41684		323
9	02:49:21.251	UPLOAD web ser...	HTTP Request	46061		323
10	02:49:21.251	UPLOAD web ser...	HTTP Request	47890		323
11	02:49:56.495	UPLOAD web ser...	HTTP Request	37519		323
12	02:50:00.097	UPLOAD web ser...	HTTP Request	38340		323
13	02:49:53.133	UPLOAD web ser...	HTTP Request	45831		323
14	02:50:02.916	UPLOAD web ser...	HTTP Request	36468		323
15	02:49:58.200	UPLOAD web ser...	HTTP Request	41370		323
16	02:50:09.142	UPLOAD web ser...	HTTP Request	30912		323
17	02:50:00.672	UPLOAD web ser...	HTTP Request	40653		323
18	02:49:56.325	UPLOAD web ser...	HTTP Request	45356		323
19	02:50:01.984	UPLOAD web ser...	HTTP Request	39729		323
20	02:50:07.313	UPLOAD web ser...	HTTP Request	34681		323

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP Requ...	20	39234	38842	45831	30912	47890	0,00%	14,9/min	,1
TOTAL	20	39234	38842	45831	30912	47890	0,00%	14,9/min	,1

Figura 38: Teste 6, serviço web de upload

Análise aos resultados para o serviço web de Upload:

Os testes efectuados ao serviço web de Upload encontram-se documentados pela sequência de imagens entre a *Figura 33* e a *Figura 38.*

Para os testes a este serviço foram utilizados dois ficheiros de dimensões bastante distintas, para serem enviados para o Repositório do Epidemic Marketplace. Para além do envio de ficheiros, este teste difere dos dois anteriores pelo facto de o método HTTP utilizado não ser o GET, mas sim o método HTTP POST.

O primeiro ficheiro tinha uma dimensão de aproximadamente 13Kb e os testes demonstram que o seu envio decorreu sem falhas, mesmo quando este era feito por vários utilizadores em simultâneo.

O segundo ficheiro tinha uma dimensão de aproximadamente 4,55Mb e apesar de o envio do mesmo também ter decorrido sem falhas, os tempos verificados nos testes para este ficheiro, são bem superiores aos tempos do primeiro, chegando a observar-se uma variação de cerca de 17 segundos entre a resposta mais rápida e a mais lenta ao envio do ficheiro.

No gráfico apresentado na *Figura 39* é mais uma vez visível a evolução dos tempos médios de resposta do serviço de Upload, aquando do aumento do número de requisições concretizadas em simultâneo.

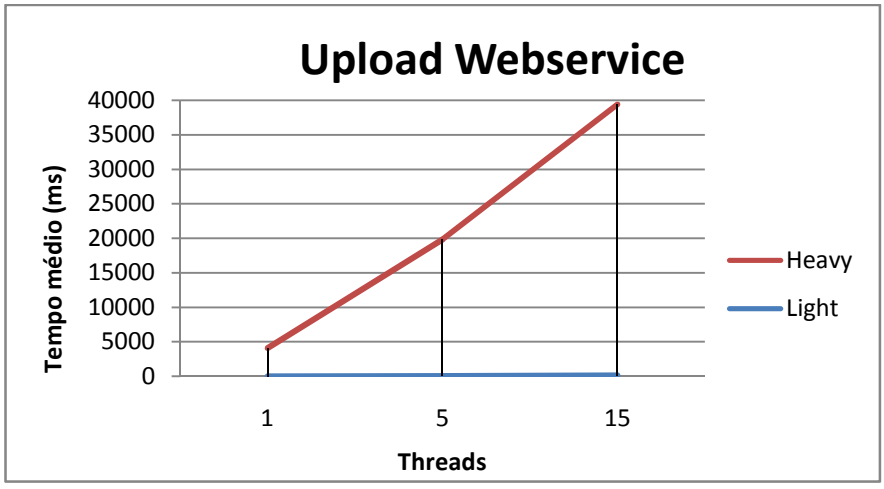


Figura 39: Gráfico de desempenho do serviço web de Download

A diferença entre o tamanho dos ficheiros utilizados nos dois diferentes testes é de tal ordem que as respostas médias dos testes com o ficheiro de menor dimensão, não são praticamente visíveis no gráfico. O aumento do tempo médio das respostas é gradual mas fica abaixo da proporcionalidade directa, quando comparado com o número de *threads* lançadas.

Análise geral dos testes realizados:

Após a realização destes testes de carga e de performance recorrendo à ferramenta JMeter, podemos concluir que os serviços disponibilizados pelo Mediador do Epidemic Marketplace, se encontram preparados para serem acedidos por vários utilizadores em simultâneo, uma vez que estes nos dão algumas garantias sobre a sua disponibilidade e capacidade de resposta.

Embora não possamos garantir a escalabilidade dos serviços para um número muito grande de acessos simultâneo (até por questões de limitação de hardware e de capacidade de processamento do servidor no qual se encontra o projecto), podemos no entanto afirmar que tendo em conta o público alvo específico a que se dirige o Mediador e o próprio Epidemic Marketplace, não são previstos problemas de indisponibilidade dos serviços, pelo menos nos primeiros anos de funcionamento do projecto Epiwork.

De notar ainda que a opção pela implementação dos serviços seguindo a metodologia REST é bastante relevante para o resultados destes testes de desempenho e de escalabilidade, uma vez que a tecnologia SOAP implica uma maior carga na requisição dos serviços web, como já foi discutido anteriormente. É então possível inferir que se os serviços web disponibilizados pelo Mediador tivessem sido construídos tendo por base a tecnologia SOAP, os tempos médios de resposta seriam superiores enquanto que a escalabilidade e disponibilidade dos serviços diminuiria.

Capítulo 6 - Conclusão

O Mediador do Epidemic Marketplace é parte integrante do projecto Europeu denominado Epiwork, o qual tem uma duração prevista de 4 anos.

O recente aparecimento de alguns surtos epidémicos à escala global, vem coincidir com uma era em que as redes sociais na internet estão cada vez mais em voga. Nestas, tornou-se comum encontrarmos referências ao estado de saúde dos seus participantes e a informações que dantes eram muito mais difíceis de reunir, como a idade, o género e a localização dos mesmos.

O Projecto Epiwork vem aproveitar a reunião destas condições para criar um inovador sistema de previsão, detecção e simulação da ocorrência de surtos epidémicos, localizados espacial e temporalmente, através não só da recolha e processamento de dados disponíveis em redes sociais, como através de dados fornecidos por outras equipas científicas envolvidas no projecto ou por parceiros institucionais.

Além da inquestionável utilidade deste inovador sistema, o projecto Epiwork fornece uma plataforma denominada Epidemic Marketplace, na qual está disponível um repositório digital onde estão armazenados um enorme conjunto de dados, devidamente categorizados e descritos através da utilização de metadados. Estes dados prevêem-se da maior importância e utilidade no trabalho dos profissionais e estudiosos da área da saúde.

A construção do Mediador da plataforma do Epidemic Marketplace visa fornecer um conjunto de serviços web responsáveis pelas operações de acesso e de escrita sobre os conteúdos digitais armazenados no Repositório da mesma plataforma. O Mediador prevê ainda a construção de uma interface RESTful que permita aos seus utilizadores uma interacção bastante simples e intuitiva e que lhes ofereça um conjunto de capacidades e funcionalidades ainda não existentes na plataforma, tornando o projecto Epiwork ainda mais interessante, inovador e atractivo.

Paralelamente à implementação dos serviços web, o Mediador fornece um esquema de autenticação baseado no método de Autorização de Acesso Básico HTTP, o qual visa garantir a segurança, integridade e confidencialidade dos dados contidos no Epidemic Marketplace.

De entre os serviços web implementados é possível destacar o serviço web de pesquisa que implementa a integração do protocolo OAI-PMH. Este protocolo encontra-se baseado num sistema de recolha de metadados que tem como objectivo permitir a partilha, difusão e armazenamento de metadados associados a recursos digitais da web, através de repositórios interoperáveis. O Mediador faz uso da *Fedora OAI Provider Interface* construindo solicitações à mesma, com os verbos definidos na sua especificação, o que lhe permite a obtenção de registos de metadados e a formulação de respostas com a sintaxe OAI-PMH. De referir que o serviço web de pesquisa permite que a mesma seja realizada sobre dois tipos de objectos digitais diferentes: Conteúdos e Colecções.

Convém ainda destacar a implementação da abordagem OAI-ORE, no serviço web de upload. Esta abordagem introduz o conceito de objectos digitais compostos e que fornece uma *framework* para a gestão e manutenção dos mesmos. Para tornar possível esta implementação e para facilitar a interacção dos utilizadores com este serviço, o Mediador disponibiliza uma aplicação cliente construída na linguagem Java, que tem a responsabilidade de realizar todo o processamento que envolve a criação de uma Mapa de Recursos no formato RDF/XML cuja sintaxe vai de encontro à da especificação da abordagem. Esta aplicação é ainda responsável pela comunicação com o Mediador, na qual é enviado o ficheiro com o Mapa de Recursos, através do método HTTP POST.

Por fim, resta falar no serviço web de download, o qual permite que os utilizadores registados no Epidemic Marketplace, dependendo do seu *role* e das suas permissões, possam realizar a obtenção de Conteúdos armazenados no repositório. Estes Conteúdos não são no entanto obtidos no seu formato binário, ao invés, uma representação em Base64 dos mesmos é embebida num ficheiro XML onde consta também a informação sobre o título original do mesmo, e posteriormente enviada para o utilizador que solicitou o serviço.

Este conjunto de serviços web são fornecidos pelo Mediador através de uma interface que segue os princípios de arquitectura REST. Como tal, uma vez que não há manutenção do contexto na comunicação, todas as solicitações aos serviços devem ser autenticadas pelo Utilizador através das suas credenciais. Esta interface RESTful permite a criação de serviços com uma sintaxe legível graças à utilização de URIs e

permite a realização de comunicações via HTTP sem geração de *overhead* (sobrecarga), uma vez que a informação que segue no pedido é apenas a necessária à execução do pedido. A utilização dos serviços acaba por se tornar bastante simples e intuitiva, uma vez que por trás destes não existe um contracto demasiadamente burocrático (como acontece nos serviços SOAP). Ao invés, o Mediador disponibiliza uma API com a descrição de cada serviço e a sua especificação.

Por fim, foi realizado um conjunto de testes aos diferentes serviços disponibilizados pelo Mediador, nos quais se pretendia avaliar o seu desempenho através do tempo de resposta e a sua escalabilidade através da simulação de utilizadores simultâneos através da funcionalidade de *multi-threading* presente na ferramenta JMeter – que se revelou muito útil embora pouco intuitiva sendo a sua aprendizagem algo demorada. Na análise final aos testes foi possível verificar que os serviços estão preparados para receber requisições enviadas de forma concorrente por diferentes utilizadores. No entanto o número de utilizadores que podem aceder aos serviços sem que estes falhem não é ilimitado e prende-se em parte com as limitações de *hardware* da máquina virtual onde se encontra alojado o Mediador. Estas limitações não são consideradas preocupantes, tendo em conta o público alvo bastante específico destes serviços e consequentemente o número possível de utilizadores.

Com o desenvolvimento do Mediador, a equipa do Work Package (WP) 3 pretende contribuir de forma ainda mais decisiva na construção e melhoramento da plataforma computacional orientada aos dados inserida no âmbito do projecto Epiwork. Em simultâneo, a criação deste Mediador, pretende fomentar a colaboração e comunicação entre o WP 3 e os WP's com os quais este está mais directamente relacionado, nomeadamente os WP 4 e 5.

6.1 Trabalho Futuro

Sendo que o Epiwork é um Projecto Europeu com a duração prevista de 4 anos e estando sensivelmente a meio desse prazo, é possível afirmar que existe espaço e tempo para que possam ser realizadas melhorias ao Mediador do Epidemic Marketplace e acrescentadas algumas funcionalidades que este ainda não possui. Um exemplo disso é a funcionalidade de remoção de Conteúdos e de Colecções existentes no repositório.

Tendo em conta o trabalho realizado, a adição de uma funcionalidade deste género serial relativamente simples.

Fora do âmbito dos serviços de mediação, o aluno considerou interessante a abordagem OAI-ORE e as potencialidades da mesma na replicação/migração de dados entre repositórios, considerando este um assunto que merece alguma atenção e aprofundamento.

6.2 Considerações Finais

A realização deste projecto e a escrita deste relatório acabaram por se revelar um grande desafio para o aluno responsável pelos mesmos. Desafio pois implicou a aprendizagem de padrões, protocolos e tecnologias com as quais nunca tinha trabalhado. Após finalizado o trabalho e olhando para trás, é possível afirmar que a curva de aprendizagem que envolveu a realização deste projecto, tem uma fase inicial muito lenta. Ao início, a percepção que se tem daquilo que se vai fazer e daquilo em que realmente consiste o projecto é muito vaga, tornando-se mais rápida à medida que se toma consciência do verdadeiro contexto e enquadramento do trabalho. Um dos erros cometidos pelo aluno na fase inicial do trabalho, esteve relacionado com a vontade de querer ver rapidamente trabalho feito e "coisas" a acontecerem. A verdade é que esta ansiedade acabou por custar caro e levou a um atraso significativo no desenvolvimento do mesmo, uma vez que foram tomadas uma série de decisões de implementação (as quais são referidas e analisadas na secção 4.6 deste relatório) que acabaram por se revelar precipitadas e que custaram muitas horas de trabalho consideradas em vão. No entanto é necessário aprender com os erros e alguns meses após o início do projecto, o aluno assentou finalmente as ideias, começou a planear e a estudar melhor aquilo que de facto tinha de fazer e quando passou para a implementação, tinha já garantias e segurança do trabalho que estava a realizar.

Numa auto-avaliação global, o aluno considera que os objectivos iniciais acabaram por ser cumpridos e que a realização deste trabalho acabou por ser uma grande lição não só ao nível prático e de implementação, mas também no modo como um projecto deste género deve ser encarado desde o início. Sem precipitações, sem pressa de ver "coisas" a acontecer, com um bom estudo inicial do problema e uma boa percepção dos objectivos.

Referências

1. *Twitter website.* [Online] Twitter. [Cited: 2010 йил 19-Julho.]

2. **Epiwork.** *Epiwork Project website.* [Online] [Cited: 2010 йил 15-Julho.] http://www.epiwork.eu/.

3. **Epidemic Marketplace.** *Epidemic Marketplace website.* [Online] [Cited: 2010 йил 10-Agosto.] http://epiwork.di.fc.ul.pt/.

4. LASIGE - Large-Scale Informatics Systems Laboratory. *LASIGE website.* [Online] Faculdade de Ciências da Universidade de Lisboa. [Cited: 2010 йил 16-Agosto.] http://lasige.di.fc.ul.pt/Main_Page.

5. *Meta-model Initial Speficiation, Catalogue of Relevant Data, Platform Requirements.* **Silva, Fabricio; Lopes, Luís F.; Couto, Francisco M.; Silva, Mário J.** Lisboa : s.n., 2009. Epiwork Deliverable 3.1.

6. *Epidemic Marketplace: an e-Science Platform for Epidemic Modelling and Analysis.* **Silva, Fabricio; Lopes, Luís F.; Couto, Francisco M.; Silva, Mário J.** Lisboa : s.n., 2009. Submitted for publication.

7. *Automated Social Network Epidemic Data Collector.* **Luis F Lopes, João M Zamite, Bruno C Tavares, Francisco M Couto, Fabrício Silva, and Mário J Silva.** 2009. Proceedings of the INForum 2009 – Simpósio de Informática.

8. **Lagoze C, Payette S, Shin E, Wilper C.** Fedora: an Architecture for Complex Objects and their Relationships. *International Journal on Digital Libraries.* 2, 2006 йил, Vol. 6, pp. 124-138.

9. **Fielding, R. T.** Architectural styles and the design of network-based software architectures. 2000 йил. PhD Dissertation. Dept. of Information and Computer Science, University of California, Irvine.

10. **Lagoze, Carl, Herbert Van de Sompel, Michael Nelson, and Simeon Warner.** The Open Archives Initiative Protocol for Metadata Harvesting, version 2.0. *Open Archives Initiative website.* [Online] Open Archives Initiative, 2002 йил. [Cited: 2010 йил 15-Agosto.] http://www.openarchives.org/OAI/openarchivesprotocol.html.

11. *Using OAI-ORE to Transform Digital Repositories into Interoperable Storage and Services Applications.* **Tarrant, D. and O'Steen, B. and Brody, T. and Hitchcock, S. and Jefferies, N. and Carr, L.** 2009 йил, Code4Lib Journal, Vol. VI.

12. *LDAP: Framework, practices, and trends.* **Koutsonikola, V. and Vakali, A.** s.l. : IEEE, 2004 йил, Internet Computing, IEEE, Vol. 8. Number 5, Pages 66-72.

13. *Epidemic Marketplace: An Information Management System for Epidemiological Data.* **Luis Filipe Lopes, Fabrício A.B. Silva, Francisco Couto, João Zamite, Hugo Ferreira, Carla Sousa, Mário J. Silva.** 2010. Proceedings of the ITBAM – DEXA.

14. *Load Testing your Applications with Apache JMeter.* **Hansen, Keld H.** Tutorial disponível em http://javaboutique.internet.com/tutorials/JMeter/.

15. *RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision.* **Pautasso, Cesare; Zimmermann, Olaf; Leymann, Frank.** Beijing - China : s.n., April 2008. 17th International World Wide Web Conference.

16. **Richardson, Leonard; Ruby, Sam.** *RESTful Web Services.* s.l. : O'Reilly, May 2007. ISBN 978-0-596-52926-0.

17. *Phishing - Wikipedia website.* [Online] Wikipedia. [Cited: 2010 йил 26-Julho.] http://pt.wikipedia.org/wiki/Phishing.

18. **Forsberg, D.** RESTful Security. 2009 йил. Helsinki: Nokia Research Center.

19. *FOAF+SSL: RESTful Authentication for the Social Web.* **Story, H., Harbulot, B., Jacobi, I., Jones, M.** Heraklion, Greece : s.n., 2009. European Semantic Web Conference, Workshop: SPOT2009.

20. *OpenID 2.0: a platform for user-centric identity management.* **RECORDON, D., AND REED, D.** s.l. : Digital Identity Management, 2006. pp. 11-16. A. Juels, M. Winslett, and A. Goto, Eds., ACM.

21. *OpenID - Wikipedia website.* [Online] Wikipedia. [Cited: 2010 йил 15-Agosto.] http://pt.wikipedia.org/wiki/OpenID.

22. *Putting the Web Services Specifications to Rest.* **III, D. R. Olsen.** Brigham : s.n., April 2008. Department of Computer Science, Brigham Young University.

23. **M. Simhachalam, R. Palkovic.** *Securing REST Web Services With Oauth.* s.l. : SUN, August 2009 йил. http://developers.sun.com/identity/reference/techart/restwebservices.html#resources.

24. *Amazon Simple Storage Service (Amazon S3).* [Online] Amazon. [Cited: 2010 йил 19-Julho.] http://aws.amazon.com/s3/.

25. *Amazon Simple Storage Service, Authenticating REST Requests.* [Online] [Cited: 2010 йил 19-Julho.] http://docs.amazonwebservices.com/AmazonS3/2006-03-01/index.html?RESTAuthentication.html.

26. **H. Krawczyk, M. Bellare and R. Canetti.** HMAC: Keyed-Hashing for Message Authentication. February 1997 йил. RFC 2104.

27. *HTTP Authentication: Basic and Digest Access Authentication.* **Franks, J. and Hallam-Baker, P. and Hostetler, J. and Lawrence, S. and Leach, P. and Luotonene, A. and Stewart, L.** 1999 йил June, Internet RFC 2617.

28. *The open archives initiative: building a low-barrier interoperability framework.* **Carl Lagoze , Herbert Van de Sompel.** January 2001. Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries. pp. 54-62. January 2001, Roanoke, Virginia, United States [doi>10.1145/379437.379449].

29. **Lagoze, C., et. al.** *Open Archives Initiative Object Reuse and Exchange, ORE User Guide – Primer.* [Online] 2008 йил 17-Outubro. [Cited: 2010 йил 22-Julho.] http://www.openarchives.org/ore/1.0/primer.html.

30. *Adding OAI-ORE Support to Repository Platforms.* **A. Maslov, A. Mikeal, S. Phillips, J. Leggett, and M. McFarland.** Atlanta, Georgia : s.n., 2009. Proc. of the 4th Intl. Conference on Open Repositories (OR'09).

31. **Dublin Core Metadata Initiative.** *Dublin Core Metadata Initiative website.* [Online] [Cited: 2010 йил 15-July.] http://dublincore.org/.

32. *Marc 21 XML Schema.* [Online] The Library of Congress. [Cited: 2010 йил 4-Agosto.] http://www.loc.gov/standards/marcxml/.

33. **Van de Sompel, H., et al.** Resource Harvesting within the OAI-PMH Framework. *D-Lib Magazine.* 12, December 2004 йил, Vol. 10.

34. **Smith, M., Bass, M., McClellan, G., Tansley, R., Barton, M., Branschofsky, M., Stuve, D., Walker, J.H.** DSpace: an open source dynamic digital repository. *D-Lib Mag.* 2003 йил, Vol. 9, 1.

35. *OpenLDAP website.* [Online] OpenLDAP. [Cited: 2010 йил 23-Julho.] http://www.openldap.org.

36. *JAX-RS: Java™ API for RESTful Web Services.* **Hadley, M. and Sandoz, P.** 2009 йил, JSR, JCP, September.

37. *Java Server Pages.* [Online] Sun. [Cited: 2010 йил 15-July.] http://java.sun.com/products/jsp.

38. *Netbeans Website.* [Online] Netbeans. [Cited: 2010 йил 26-Julho.] http://www.netbeans.org.

ANEXO I – Client Application to the Upload Functionality in the Epidemic Marketplace Mediator:

Introduction:

The Mediator from the Epidemic Marketplace platform will provide its users a set of web services that will be responsible for liaison between the several components of the platform and external components to it.

Among the web services for which the mediator is responsible, is building the functionality to upload content to the digital repository existing Epidemic Marketplace. The implementation of this functionality shall conform to the standards described in the approach OAI-ORE (Open Archives Initiative - Object Reuse and Exchange), a standard for description and exchange of aggregations of web resources. This implementation should both be in accordance with the methodology for the development of web systems defined by the principles of REST architecture.

The OAI-ORE approach introduces the notion of a map of resources represented by a RDF (Resource Description Framework) graph, which describes an aggregation, the aggregate of resources that it contains and the relations between them. The RDF graph representing the map of resources can, however, be serialized in any RDF syntax.

As such, the OAI-ORE project proposes to use the syntax RDF / XML serialization of these same maps.

The union of these criteria and conditions means that the transaction in which users Upload contents to the repository in the Epidemic Marketplace, consists in sending a file that meets the RDF / XML syntax and standards defined by the OAI-ORE approach.

This file represents the aggregation and it shall contain a description of the aggregation, of the resources that this involves, of the relationship between resources

and of the metadata according to the Dublin Core standard, associated with its resources.

In the solution for the implementation of this feature it was chosen that the resources to be stored in the repository, should be forwarded through their inclusion in RDF / XML files sent to the mediator by users.

The Mediator of the Epidemic Marketplace aims to provide a set of innovative web services that make the interaction between users and the repository of the platform, easy, intuitive and simple. Delegate in the users the construction of the RDF / XML file that meets all the criteria and standards described above, leads to a contradiction in the objectives for the operation of the Mediator, taking into account the complexity that this process entails.

The construction of an HTTP POST request where the file is sent to the mediator, isn't also a trivial task and requires some programming knowledge if the user wishes to accomplish the send of the file through the web service upload, instead of using the interface restfull to do so.

Thus, the chosen solution involves the creation of a Java class (UploadClient.java) that is available to the authorized users of the Epidemic Marketplace, enabling them to automatically generate the RDF / XML file by calling the constructor method of the class, which receives a set of parameters, provided by the user, that are necessary to construct File.

Subsequently, the file generated by the class is automatically sent to the Mediator, through the call of some methods of a helper class, which is responsible for building the HTTP POST request where the file is included.

Below, will be presented information in greater detail in connection with the operation of the UploadClient class.

Primary Methods

In this section will be presented the methods with which the users will directly interact.

Constructor Summary:

UploadClient(String user, String pass, String collection, ArrayList<Object> data, ArrayList<File> metadata)

Generates a new RDF / XML file, in which will be included the objects that are contained in the Object ArrayList “data”, whether they are Files or URLs and that are associated with the metadata files contained in the File ArrayList “metadata”.

The String “collection” is also included in the RDF/XML file and describes the title of the Collection in the repository where the contents will be stored.

The file is then sent to the Mediator along with the user credentials present in the “user” and “pass” Strings.

Constructor Detail:

public UploadClient(String user, String pass, String collection, ArrayList<Object> data, ArrayList<File> metadata) throws IOException, TransformerException

Generates a new RDF / XML file and sends it to the Mediator.

Parameters:

user – the username of the Epidemic Marketplace authorized user

pass – the password of the Epidemic Marketplace authorized user

collection – the title of the collection where the contents will be stored

data – The data (Files/URLs) to be stored in the repository

metadata – the metadata that describes the data to be stored

Throws:

IOException – Notes the occurrence of an exception of I / O.

TransformerException – Specification of an exceptional condition that can occur during the transformation process.

Secundary Methods

This section presents the methods with which users will not interact directly, but that are part of the Class and are essential for it proper processing.

This methods will also help users in the event they wish to rebuild this program in any other programming language.

Method Summary:

public void doUpload(File file)

Method that performs the Upload of the File received as an argument which represents the file with RDF / XML syntax generated by this class, to the Mediator of the Epidemic Marketplace.

The upload is performed through the call to the methods within ClientHttpRequest.java class that has the responsibility to build the HTTP POST request where the file is sent.

String treatName(String fName)

Performs treatment of the String received as an argument, replacing space characters with "_" in order to avoid errors in the validation of the syntax of the RDF / XML.

String uniqueFileName()

Generates a unique file name, through a time stamp which will be assigned to the RDF / XML file.

Document makeDoc(Object obj)

Creates a new Document through the user information passed to the constructor method.

The Object element received may be represented for a File or a URL, which will influence the processing method.

Document describeFileObjs(Document doc)

Method executed in the case that the user passes a File as an argument in the call to the constructor method.

Change the Document received, returning a new Document.

Document describeUrlObjs(Document doc)

Method executed in the case that the user passes a URL as an argument in the call to the constructor method.

Change the Document received, returning a new Document.

Document describeAgreg(Document doc)

Method that handles the Document received as an argument and changes it, by adding information about the RDF / XML file, as defined in the OAI-ORE approach.

void writeXmlFile(Document doc, File fich)

Method that handles the Document received as an argument, generating a new file with the RDF / XML syntax.

byte[] getBytesFromFile(File file)

Method that transforms the File received as argument, in a byte array.

byte[] encode(File binFile)

Method that encodes the File received as an argument, in an array of bytes with Base64 encoding.

Method Detail:

public void doUpload(File file) throws MalformedURLException, IOException

Method that performs the Upload of the File received as an argument which represents the file with RDF / XML syntax generated by this class, to the Mediator of the Epidemic Marketplace.

The upload is performed through the call to the methods within ClientHttpRequest.java class that has the responsibility to build the HTTP POST request where the file is sent.

Parameters:

file – the RDF/XML file generated by this class.

Throws:

IOException – Notes the occurrence of an exception of I / O.

MalformedURLException - Thrown to indicate that a malformed URL has occurred.

private String treatName(String fName)

Performs treatment of the String received as an argument, replacing space characters with "_" in order to avoid errors in the validation of the RDF/XML syntax.

Parameters:

fName – the original name of the file that represents the resource.

Returns:

A String with the name of the resource appropriately modified.

private String uniqueFileName()

Generates a unique file name, through a time stamp, which will be assigned to the RDF / XML file.

Returns:

A String representing the filename of the RDF / XML file generated.

public Document makeDoc(Object obj)

Generates a new Document.

Parameters:

obj – the Object passed by the user that can be a File or a URL.

Returns:

A new Document that will later be transformed into a file with a RDF / XML syntax.

public Document descreveFileObjs(Document doc)

Add to Document received as a parameter, information on resources that are part of the aggregation.

Parameters:

doc – The Document to be modified.

Returns:

A Document with information about the resources that are part of the aggregation.

public Document descreveUrlObjs(Document doc)

Add to Document received as a parameter, information on resources that are part of the aggregation.

Parameters:

doc – The Document to be modified.

Returns:

A Document with information about the resources that are part of the aggregation.

public Document descreveAgreg(Document doc)

Add to Document received as a parameter, information on the aggregation.

Parameters:

doc – The Document to be modified.

Returns:

A Document with information on the aggregation.

private void writeXmlFile(Document doc, File fich) throws TransformerException

Method that processes the received Document as an argument, generating a new file with the RDF / XML syntax.

Parameters:

doc – the Document that will be used to build the RDF / XML file.

fich – the RDF/XML file.

Throws:

TransformerException – Specification of an exceptional condition that can occur during the transformation process.

private byte[] getBytesFromFile(File file) throws IOException

Method that transforms the File received as an argument, in an array of bytes.

Parameters:

file – the File wanted to be stored in the repository of the Epidemic Marketplace.

Returns:

An array of bytes representing the file content.

Throws:

IOException – Notes the occurrence of an exception of I / O.

private byte[] encode(File binFile) throws IOException

Method that encodes the File received as an argument, in an array of bytes with base64 encoding.

Parameters:

file – the File wanted to be stored in the repository of the Epidemic Marketplace.

Returns:

An array of bytes representing the file content.

Throws:

IOException – Notes the occurrence of an exception of I / O.

Example of use of the UploadClient.java Class

It is possible to test the class UploadClient through a simple Java program that creates a new instance of the class.

In the test program below, one instance of the class UploadClient is created.

The “data” ArrayList represent the Objects that the user wants to store in the repository. These objects can be either Files or URLs. If the object represents a file, this file will be encoded in base64 and then its codification is included into the RDF / XML generated. Each object is described by one metadata file in the xml format, which is contained in the “metadataFiles” ArrayList. The String “collection” it’s also included into the RDF/XML file and represents the title of the collection in the repository, where the contents will be stored.

The instantiation of the UploadClient class presented, result in the generation of a RDF / XML file on the user local machine, which is subsequently forwarded to the Mediator through an HTTP POST request made to the web service of uploading contents. Along with this file, the user credentials for the Epidemic Marketplace are sent in the request.

****

```
import java.io.File;
import java.io.IOException;
import java.net.URL;
import javax.xml.transform.TransformerException;

public class TesteUploadClient {

  private static ArrayList<Object> data;
  private static ArrayList<File> metadataFiles;
```

```
  public static void main (String[] args) throws IOException,
TransformerException{

       data = new ArrayList<Object>();
       metadataFiles = new ArrayList<File>();
       String username = "hmferreira";
       String password = "12345";
       String collection = "Test Collection";

            File file1 = new File("/home/hmferreira/repositories.pdf");
            URL url1 = new URL("http://exemplo.mediator/idfile=10");
            data.add(file1);
            data.add(url1);

            File metadataFile1 = new File
            ("/home/hmferreira/fileMetadata.xml");
            File metadataUrl1 = new File
            ("/home/hmferreira/urlMetadata.xml");
            metadataFiles.add(metadataFile1);
            metadataFiles.add(metadataUrl1);

            UploadClient uc = new UploadClient(username, password,
            collection, data, metadataFiles);

  }
}
**********************************************************
```

ANEXO II – Ligação HTTPS entre Mediador e LDAP

De modo a garantir que a comunicação entre o Mediador e os seus utilizadores, assim como a comunicação entre o Mediador e as outras aplicações web de que este necessita para funcionar da forma prevista, fossem comunicações seguras sem perda da integridade e confidencialidade dos dados, optou-se por utilizar o protocolo HTTPS (HyperText Transfer Protocol Secure) em combinação com o protocolo criptográfico SSL/TLS (Secure Sockets Layer / Transport Layer Security).

Foi inicialmente necessário possibilitar a utilização do protocolo HTTPS no servidor web Tomcat sobre o qual o Mediador realiza as suas operações, para tal procedeu-se à criação de um keystore certificado, utilizando a ferramenta "keytool". Posteriormente, num dos ficheiros de configuração do servidor Tomcat, foi necessário realizar algumas alterações para que o mesmo ficasse apto a receber ligações na porta 443, que por defeito é a porta definida para escutar ligações SSL.

Após estas configurações é então possível realizar conexões ao servidor Tomcat via HTTPS.

Para permitir ligações SSL/TLS ao Tomcat, um certificado do servidor é requerido pelos protocolos SSL/TLS. Para efectuar a criação dos certificados foi utilizada a ferramenta OpenSSL.

Através do *script* CA da ferramenta OpenSSL criou-se então um certificado de raíz para o servidor Tomcat, reconhecido por uma autoridade certificadora (CA - Certificate Authority ou Certification Authority).

A autoridade certificadora pode emitir vários certificados, na forma de uma estrutura em árvore. No entanto, o certificado raíz é hierarquicamente o certificado mais importante desta árvore, uma vez que funciona como chave primária necessária para assinar os demais certificados.

De seguida e uma vez mais recorrendo ao uso do OpenSSL, para efectuar a criação de um Pedido de Assinatura do Certificado (CSR – Certificate Signing Request), desta feita relativo ao servidor Ldap. Este pedido consiste numa mensagem enviada a partir de um candidato, a uma autoridade de certificação, com o intuito de solicitar um certificado de identidade digital.

O Pedido de Assinatura do Certificado é então satisfeito pela Autoridade Certificadora, através da sua chave privada resultando deste processo um certificado para o servidor OpenLdap devidamente assinado pela CA. Neste ponto já nos encontramos na posse dos certificados que são precisos para o funcionamento do servidor OpenLdap, é agora necessário realizar as configurações inerentes à utilização dos certificados por parte do servidor.

Nos ficheiros de configuração do Ldap, foram realizadas as acções de configuração necessárias e indicados os caminhos para os certificados entretanto criados. Estas alterações possibilitaram a utilização do protocolo LDAPS (não standard), no porto 636 que é o porto atribuído por defeito às comunicações que utilizam o protocolo LDAP sobre a tecnologia SSL/TLS.

Estes passos deveriam ser suficientes para a utilização do protocolo LDAPS e consequentemente para a autenticação de forma segura, dos utilizadores do Epidemic Marketplace. No entanto, uma vez que o certificado que estamos a utilizar é considerado um certificado de teste, por ter sido criado através da ferramenta keytool em vez de ser originário de uma Autoridade de Certificação comercialmente bem conhecida, como por exemplo a “Verisign” ou a “GoDaddy” e uma vez que estamos a tentar estabelecer uma ligação SSL com um *host* através de JSSE, é lançada a seguinte excepção:

> *“javax.net.ssl.SSLHandshakeException:*
> *sun.security.validator.ValidatorException:*
> *PKIX path building failed:*
> *sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target when trying to open an SSL connection to a host using JSSE. “*

Esta situação pode ser ultrapassada através da utilização de um programa em Java, que estabeleça uma ligação com o servidor pretendido e inicie o processo de *SSL Handshake,* processo que ocorre obrigatoriamente antes do início de qualquer sessão SSL, no qual o servidor se autentica perante o cliente permitindo depois que ambos

colaborem na criação de chaves simétricas, utilizadas para encriptação, decriptação e detecção de tentativas de sabotagem durante a sessão SSL que se segue.

O programa utilizado para ultrapassar esta situação foi o InstallCert.java, que basicamente estabelece um ligação com o servidor pretendido, inicia o processo SSL Handshake, imprime a declaração da excepção lançada e revela o certificado utilizado pelo servidor, dando ao utilizador a possibilidade de adicionar o mesmo à sua KeyStore confiável. Ao fazê-lo, o problema desaparece e é então possível estabelecer uma ligação SSL com o servidor.

O que no caso do Mediador se traduz na possibilidade de autenticar os seus utilizadores de forma segura, através da utilização do protocolo LDAPS.